EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

HOS / SEQUENCER

User Manual

Doc.No. VLT-MAN-ESO-17220-0737

Issue 3

Date 28/03/2002

Prepared E. Allaert 28/03/2002 ....

Name Date Signature

Approved G. Raffi ....

Name Date Signature

Released G. Raffi ....

Name Date Signature

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0	17/11/94	All	First version
1.1	02/03/95	Section 2	Release of Sequencer version 1.2 with new functionality
1.2	30/07/95	Sections 2 and 4	Release of Sequencer version 1.11 with new functionality Distinction between Sequencer shells and Sequencer tool
1.3	15/01/96	Section 2	Some commands added for release 1.21 of Sequencer shells
1.4	28/05/96	Sections 2, 3	Reflected new version of Tcl/Tk, presence of RTD and Msqltcl extensions; also 2 new commands
1.5	24/04/97	Sections 2, 3	Reflected new version of Tcl/Tk Added section 2.4.4 (SPR 970101) Added section 2.6 (dynamic loading) 2 new commands
2.0	07/11/1997	Sections 2, 3, 4	Added description of new commands/extensions in sections 2.3, 3 and 4. Added section 2.8 (Internet resources on Tcl/Tk)
2.1	13/10/1998	Section 2.3.2 Sections 2.3.8, 2.7 Section 2.3.10	Added obituaries (SPR 960459) New var. *seq_errReplyMerging* (SPR 980464) New command *seq_deleteHandle*
2.2	06/03/2001	Section 2.3 Section 2.3.11 Section 2.8 Section 3	Updated version numbers of components Added *seq_debug* and *seq_redirect* description Updated list of URLs Updated version numbers/instructions.
3	28/03/2002	Section 2.3.4 Sections 2, 3.1	Added *seq_logFits** cmds (SRP 2001098) Reflects new version of Tcl/Tk

The information contained in this manual is intended to be used in the ESO VLT project by ESO and authorized external contractors only.

While every precaution has been taken in the development of the software and in the preparation of this documentation, ESO assumes no responsibility for errors or omissions, or for damage resulting from the use of the software or of the information contained herein.

1 INTRODUCTION 1
1.1 PURPOSE 1
1.2 SCOPE 1
1.3 REFERENCE DOCUMENTS 1
1.4 ABBREVIATIONS AND ACRONYMS 1
1.5 GLOSSARY 1
1.6 STYLISTIC CONVENTIONS 2
1.7 PROBLEM REPORTING / CHANGE REQUEST 2

2 USER'S GUIDE 3
2.1 OVERVIEW 3
2.2 SCRIPTING LANGUAGE SYNTAX 4
2.2.1 Basic rules 4
2.2.2 Substitution 4
2.2.3 Quoting 5
2.2.4 Potential pitfalls 5
2.3 SEQUENCER SHELL COMPONENTS 7
2.3.1 CCS Environment Interface 8
2.3.2 CCS Message System Interface 8
2.3.3 CCS On-line Database Interface 10
2.3.4 CCS Logging System Interface 10
2.3.5 CCS Error System Interface 11
2.3.6 CCS Scan System Interface 11
2.3.7 CCS Event Handling 11
2.3.8 Error handling 12
2.3.9 OSLX 12
2.3.10 Convenience procedures 12
2.3.11 Debugging help 13
2.4 RUNNING THE SEQUENCER SHELLS 14
2.4.1 Before you start 14
2.4.2 Starting up 16
2.4.3 Example Sequencer script 18
2.4.4 Subprocesses 20
2.5 THE SEQUENCER AS AN IMBEDDABLE INTERPRETER 21
2.6 DYNAMICALLY LOADABLE EXTENSIONS 21
2.6.1 Initialization of a loadable extension 21
2.6.2 Compiling loadable extensions 22
2.6.3 Loading a package 22
2.6.4 Examples from the Sequencer 23
2.7 SEQUENCER VARIABLES 23
2.8 INTERNET RESOURCES ON TCL / TK 25

3 INSTALLATION 27
3.1 REQUIREMENTS 27

3.2 COMPILATION AND INSTALLATION OF THE SEQUENCER 28
3.3 VERIFICATION 29

4 REFERENCE 31

5 PUBLIC INTERFACE FILE 149

1 INTRODUCTION

1.1 PURPOSE

This document is the User Manual for the HOS/Sequencer version 2.69 and is intended to provide all the necessary information for the installation and use of this module.

1.2 SCOPE

The Sequencer-module in its present release (2.69) is limited to an embeddable interpreter with low level commands. The use of this interpreter and its shells (*seqWish* and *seqSh*) will typically be in workstation applications offering higher level functionality, including a user interface and acces to the CCS functionality.

The user-interface to create and execute Sequencer-scripts is named *sequencer*. It is currently not included. It will create scripts that can be interpreted by *seqWish* and *seqSh*, and will itself be a set of Sequencer-scripts.

1.3 REFERENCE DOCUMENTS

The following documents are referenced in this document.

[1] Tcl and the Tk toolkit, John K. Ousterhout, ISBN 0-201-6337-X

[2] Practical Programming in Tcl and Tk, Brent Welch, ISBN 0-13-182007-9

[3] VLT-MAN-ESO-17200-0981, 1.0 15/01/96 -- VLT SW Problem Report Change Request User Mnl

[4] VLT-MAN-ESO-17229-0866, 2.8, 16/05/1999 -- VLT SW Real Time Display User Manual

1.4 ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used in this document::

CCS	Central Control Software
HOS	High Level Operating Software
HW	Hardware
I/O	Input/Output
SW	Software
TBD	To Be Defined
VLT	Very Large Telescope
WS	Workstation

1.5 GLOSSARY

sequence

a set of commands in Sequencer language, generally intended to define and execute a series of related observations. These sequences are to be interpreted by a sequencer shell.

Tcl/Tk

Tool **c**ommand **l**anguage / **T**ool**k**it. A general purpose scripting language designed and implemented by Dr. John Ousterhout of the University of Berkeley, CA (presently working at Sun Microsystems Laboratories).

1.6 STYLISTIC CONVENTIONS

The following styles are used:

bold
in the text, for commands, filenames, pre / suffixes as they have to be typed.

italic
in the text, for parts that have to be substituted with the real content before typing.

`teletype`
for examples.

`<name>`
in the examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

1.7 PROBLEM REPORTING / CHANGE REQUEST

Please refer to [3]:

1. to report a problem encountered using the software and / or documentation
2. to suggest changes in the software or documentation

2 USER'S GUIDE

2.1 OVERVIEW

The Sequencer is intended as a general purpose tool, allowing both technical and scientific staff to ease the definition and execution of sequences of commands, intended to control the telescopes and instruments.

The Sequencer is based on Tcl/Tk. This is an embeddable interpreter, with a quite powerful core allowing a.o. control structures and arithmetic operations. Tcl/Tk was first released as free software in early 1990, and since then the number of its users has grown exponentially. The amount of good quality add-ons to Tcl/Tk available on the Internet is constantly growing, and several of the extensions find their way into the Tcl/Tk core every time a new version is released. The current Internet home site for Tcl/Tk is SourceForge (*www.sourceforge.net/project/tcl*). A good WWW starting point for more information about Tcl/Tk is the *Tcl'ers WIKI* (URL *http://mini.net/cgi-bin/wikit/name).* More references to Tcl/Tk information available via the Internet are given in section 2.8.

The Sequencer in its present version is just another of these extensions incorporated into the Tcl/Tk core. It adds a series of commands which give access to CCS facilities. Using these commands as basic building blocks, combining them together into procedures (*scripts*), one can easily add higher level commands. Compared with standard programming languages as C, small scripts are most of the time easier to write and debug, and the code is much more compact (hence maintainable). It is obvious that for many applications the Sequencer will be a handy prototyping tool. And even in some cases there may be no need to convert this Sequencer-language prototype to C-language, in particular whenever performance improvements are irrelevant.

The present release of the Sequencer contains 2 applications with this interpreter. These shells address mainly the needs of technically oriented people, and will not satisfy the generic public. The reason is quite simple: being an extension to Tcl/Tk, the Sequencer's syntax is the one inherited from Tcl/Tk; such syntax - as the syntax of any popular programming or shell language - will not be mastered or even appreciated by many observers. It simply does not allow to describe easily and intuitively the steps required to execute a *sequence of observations*.

The Sequencer needs therefore to be extended towards the higher level. There are a couple of directions which can offer solutions:

- The development of higher level commands; such procedures can hide to some extent the intricacies of the system and the scripting language - at least if these higher level commands do not allow lots of arguments and/or options.
- The use of tables; tables describing sequences of repeated actions are easy to construct on any hardware platform, very intuitive, and straightforward to interpret by an interpreter like the Sequencer. This can be considered a specific case of the previous point.
- The use of visual programming tools; this area looks quite promising. If carefully designed and implemented, it gives a means to "write" and execute sequences in a very intuitive way. This is being investigated right now.

Careful readers will have noticed that technically speaking the Sequencer is more than one application. First of all we have the *interpreter-shells*, able to interpret existing scripts, and secondly there is (or will be) a *tool* to assist in the creation of sequencer-scripts. Although in principle the tool can be used by anybody (provided there are different user levels), writing a script with an editor will in most cases outperform the script created with the tool.

The two shells that are part of the Sequencer module are called *seqSh* and *seqWish*. The latter includes the usual Tk widget commands, while the former does not have any Tk / windowing capabilities or commands and consequently does not require a DISPLAY environment variable. The name *sequencer* as application name is reserved for the tool described above.

With the integration of the Sequencer module into one of the latest Tcl cores, dynamic loading became available[1]. This means that extensions with extra functionality can be loaded during runtime, if they are built according to a certain scheme. There is no longer the need to create a monolithic interpreter containing all the extensions you may ever need: they can be loaded into memory as the need presents itself. Some of the functionality of the Sequencer is offered this way. A detailed explanation of this feature is given in section 2.6.

2.2 SCRIPTING LANGUAGE SYNTAX

The Tcl and Sequencer shells[2] language have only a few basic rules. A complete description can be found in [1], chapter 2, and in [2]. This is summarized in the rest of this section, paying special attention to potential traps and pitfalls.

2.2.1 Basic rules

The Sequencer shells interpret *commandlines*. These commandlines can be given interactively, one by one, or they can be grouped together in a *script*. A script is a collection of one or more commandlines, often grouped together in a file. The basic syntax for all Sequencer shell commandlines is:

```
command arg1 arg2 arg3 ...
```

So each commandline consists of one or more *words*, whereby words are separated by spaces or blanks. The first word of a commandline is the *command*, which is either the name of a built-in command (programmed in C) or a Sequencer shell procedure (programmed in Tcl / Sequencer). The following words are the *arguments* passed to that command. Arguments are string-valued, i.e. they are always passed as strings to the command.

The commandline is terminated by a newline or a semicolon. All commandlines return a string as a result, which in some cases (depending on the command and its arguments) can be an empty string. This exposes one of the fundamentals of Tcl: everything is a string.

On top of this simple piece of syntax there are only the rules for *substitution* and *quoting*. These are the only mechanisms employed by the Tcl interpreter before it runs a command. After single pass substitution the arguments are, as said, passed as strings to the command, which has to interpret them. Features like control flow, procedures, and expressions are implemented as *commands* rather than *grammar*. They are not understood directly by the Tcl interpreter.

2.2.2 Substitution

The Tcl interpreter can perform 3 different types of substitution :

1. **variable substitution**: when a variable has been set previously, it can be de-referenced by using a $-sign in front of the variable name. The Tcl interpreter replaces in the commandline the occurrences of the $-sign followed by a variable name with the string-value of this variable. Example:

```
set myVar 53; expr $myVar*$myVar
```

[1] Dynamic loading has been supported in Tcl since the release of Tcl 7.5

[2] As far as syntax is concerned, Tcl and the Sequencer are identical, and are used interchangeably in this context.

will associate the string `53` with variable name `myVar`, and then calculate the square of the variable `myVar`. It is equivalent to

```
expr 53*53
```

2. **command substitution**: this allows to use the result of one commandline as an argument in another commandline. Command substitution is invoked by square brackets. Nesting is allowed.
   Example:

   ```
   set c [expr $myVar*$myVar]
   ```

   makes the Tcl interpreter first evaluate everything inside the brackets, and that result (the string `2809`) is used as the second argument for the `set` command.

3. **backslash substitution**: the backslash character has in fact two functions. It is an escape character for the previous substitutions and also allows to use various unprintable characters like newline in a commandline. Example:

   ```
   set squareBrackets \[\]; set doubleQuote \”
   set newLine \n
   ```

2.2.3 Quoting

In 2.2.1 we have seen that blanks are separators for Tcl words. This leads to a problem when a single argument is or should be a string containing blanks. We need something that allows to group words[1] together. There are actually two ways to obtain that.

1. **Double quotes**. Blanks and even newlines within double quotes are part of the Tcl word. Substitutions as described in 2.2.2 still take place. Double quotes do not nest, as the opening and closing double quote are the same ASCII character.
   Example:

   ```
   set result “The square of $myVar is [expr $myVar*$myVar]”
   puts stdout $result
   ```

   will print the string `The square of 53 is 2809` to stdout.

2. **Curly braces**. The only character that is special after an opening curly brace is a closing curly brace. All other characters, including blanks, newlines and double quotes have no special meaning and are part of the Tcl word. Substitutions are prevented within matching curly braces. Braces nest.
   Example:

   ```
   set result {The square of $myVar is [expr $myVar*$myVar]}
   puts stdout $result
   ```

   will print the string `The square of $myVar is [expr $myVar*$myVar]` to stdout.

2.2.4 Potential pitfalls

Although the rules for the Tcl syntax are simple and few, many people experience some difficulties with it in the beginning, mainly because the behaviour is different from what one is used to in another language. So take care about the following!

- The grouping of words by quoting is done *before* substitutions are performed. I.e. the values of variables or command-results do not affect grouping.
  Example:

[1] Remark that from this point on there is a difference between *normal* words and *Tcl* words: the former cannot contain whitespace or newlines, the latter is like any C string.

```
        set a \[expr
        set b \]
        set c 50
        puts stdout “The square of c is $a $c*$c $b”
```

will print the string *The square of c is [expr 50*50]*

- There is only a single round of substitutions before command invocation. I.e. the result of a substitution is not interpreted a second time.
  Example:

  ```
          set a 50
          set b \$a
          set c $b
  ```

  will set `c` to the string *$a*, not to the string *50*.
- A left curly brace can only be preceded by whitespace (word separator), another left brace (nesting) or opening square bracket (command substitution). Likewise, a right curly brace can only be followed by whitespace, another right brace or closing square bracket. The Tcl interpreter will trip over and complain about any other character.
- A double quote character can only be an *opening* quote for grouping if it is preceded by whitespace.
  Example:

  ```
          set tenInch 10”
  ```

  is perfectly legal and will set `tenInch` to the string *10”*.
- Square brackets used for command substitution do *not* require spaces around them.
  Example:

  ```
          set a [expr 11*11][expr 12*12]
  ```

  will set `a` to the string *121144*. Remark that a space between the first closing and second opening bracket would produce a syntax error.
- For command substitution, newlines and semi-colons are significant as commandline terminators (contrary to quoting). Use the backslash substitution to escape such terminations.
  Example:

  ```
          set theNameOfThisVariableIsSoLongTheLineWillWrap [expr 12345*
          6789]
  ```

  will produce a syntax error. To avoid that, append a backslash to the end of the first line.
- Although e.g. the *set* command can manipulate arrays, the Tcl interpreter itself does not know any special data types. This means that in the case of arrays, the parsing of the round brackets is done by the command, not the Tcl interpreter.
- It is very important where you have opening braces for the arguments of all commands that can take scripts or lists as arguments (control flow commands, procedures,...). A construct like

  ```
          if {$myVar != 1}
          {
              echo “unexpected value for myVar: $myVar”
          }
  ```

  will result in an error, as the *if* command needs two arguments, and the newline after the first argument terminates the commandline. This behaviour is due to the fact that control structures are depending on *commands*, and not *grammar*, like in C. So the above example will be correct if we write it as

  ```
          if {$myVar != 1} {
  ```

```
                echo “unexpected value for myVar: $myVar”
            }
```

Remark that the space on the first line between the closing brace of the first argument and the opening brace of the second is essential.

2.3 SEQUENCER SHELL COMPONENTS

The Sequencer shells *seqSh* and *seqWish* are statically built on top of Tcl/Tk 8.3.3; seqWish, which includes Tk functionality, contains all of the following extensions[1]:

- **[incr Tcl]** version 3.2 - the extension that supports object oriented programming in Tcl/Tk. This also includes **[incr Tk]** version 3.2, which gives the framework to build megawidgets, and **iwidgets 3.0.1**, which is a set of megawidgets.
- **TclX** version 8.3 and **TkX** version 8.3 - the extended Tcl/Tk command set, adding a.o. several UNIX commands, file I/O commands and math commands
- **BLT** version 2.4u - the Bell Lab Toolkit; consists of about a dozen graphical commands, all in the *blt::* namespace
- **msqltcl** version 1.99 - an extension that provides high-level access to a Mini SQL (mSQL) database server. Mini SQL (mSQL) is a freely available, lightweight database engine.
- **Img** version 1.2.4 - a package which enhances Tk, adding support for many other Image formats: BMP, XBM, XPM, GIF (with transparency), PNG, JPEG, TIFF and postscript.

seqSh on the other hand does not have X-capabilities by default, i.e. it contains only Tcl, [incr Tcl], TclX and msqltcl. However, dynamic loading permits to add Tk, TkX, [incr Tk], iwidgets, BLT and Img at runtime into seqSh.

These listed extensions are available as *static packages*, i.e. they are linked statically into the Sequencer shells, and are as such available from the very start-up of the Sequencer shells. There is nothing extra to be done to have access to the functionality offered by these extensions. Other extensions (like *expect*) can be loaded dynamically as the need arises. For a detailed explanation of these extensions please refer to their accompanying documentation and manpages. The rest of this section will concentrate on the specifics of the Sequencer shells.

All the Sequencer shell specific extensions to Tcl show up as commands with the prefix *seq_*. In as far as these extensions are atomic and have a clear mapping to a CCS function, the rest of the command name is the one of the corresponding CCS command, with the same case (e.g. *seq_dbGetAttrInfo*). For the other commands similar naming conventions are used. Both Sequencer shells (seqSh and seqWish) include an identical set of *seq_* commands.

For all CCS functionality required in the Sequencer shells, atomic commands are made, attempting to map a single CCS function into a single Sequencer shell command. These basic commands can then be used to build more complex ones, i.e. Tcl/Sequencer *procedures*. From the user’s perspective, there is no difference between procedures and C-coded commands, except perhaps for a slight performance penalty in the case of Tcl scripts.

In what follows the different Sequencer shell commands are grouped according to functionality, and are described shortly. More detailed explanation can be found in the references (chapter 4). Re-

[1] Remark that up to and including the November 1996 release of the Sequencer, RTD was a standard part of seqWish. This is no longer the case: RTD is now a loadable extension. If any RTD function is required within seqWish, if suffices to give first the command **package require Rtd**.

mark that there are currently no differences between full (RTAP-based) CCS and CCS-light: all commands and libraries available in the former are also available for the latter. On the other hand, if the Sequencer shells are built in an environment without CCS, the CCS-calls will obviously not be part of it. One notorious exception to this is for the error handling, where a CCS-like replacement for some *err** calls exist and are being used in the Sequencer shells. In particular, *seq_errResetStack*, *seq_errAdd* and *seq_errCloseStack* exist also for a no-CCS environment.

2.3.1 CCS Environment Interface

- **seq_ccsInit**
  Maps into *ccsInit(3)*; to register the Sequencer shell with CCS as the process with an arbitrary name (<*myProcName*>). After this command has been issued, this process is known to CCS as <*myProcName*>, which is one of the parameters required to address this process via CCS (cf. *msgSendCommand(3)*).
- **seq_ccsExit**
  To execute *ccsExit(3)*, and reset internal linked lists containing information about commands with pending replies or about registered events.
- **seq_ccsAsyncInput**
  This command does not have a counterpart in CCS; it enables or disables the acceptance of asynchronous CCS messages. If enabled, and when messages come in, they will be dispatched immediately using the services provided by *seq_msgDispatch* (see 2.3.2); otherwise they will be rejected returning an error to the originator.

2.3.2 CCS Message System Interface

- **seq_msgSendCommand**
  Maps into *msgSendCommand(3)*; to send a command to a certain destination identified by the parameters of the command. If successful, this command will return a *command handle*. This handle is internally linked to the relevant information of this command, permitting a proper identification and filtering of incoming replies. Part of this information is a unique integer number which is passed as the *msgCMDID* parameter in the *msgSendCommand(3)* call, and it is therefore imperative that the recipient processes of such commands return the original command identifier in their reply. When all replies on a certain command have been dealt with, the respective command handle is automatically deleted. Checking of the CDT of the receiving process can be enabled / disabled by means of the *seq_ccsCmdCheck* variable or by using the *-(no)check* option with the command.
- **seq_msgRecvReply**
  Although without a direct one-to-one relationship with a CCS message system call, this command is similar to a *msgRecvMsg(3)* with a properly set filter. It allows to receive a single or all replies to a previously given command, which is identified by its handle (the *cmdId* returned by *seq_msgSendCommand(n)*).
  An alternative way to retrieve and deal with incoming replies is provided by the event mechanism (see 2.3.7). This allows to attach a callback to a command handle, which will be executed in the background (at global level) whenever a reply comes in.
- **seq_msgSendReply**
  Maps into *msgSendReply(3)*; it is used to return a reply to a process which sent a *SCRIPT* command to the Sequencer shell (see *seq_msgDispatch*). Normally, the Sequencer shell will return an empty reply immediately after receiving the *SCRIPT* command, and a second, final

reply after evaluating the script. This is done automatically. If intermediate replies are needed (e.g. in the *VERBOSE* mode), *seq_msgSendReply* has to be used, from within the evaluation of the script.

- **seq_msgList**
  Does not have a corresponding CCS message system call. It lists the commands for which there are still replies pending. The information returned includes the *cmdId*, the name of the command, plus the environment and process it was sent to.
- **seq_msgCheck**
  Does not have a corresponding CCS message system call. It is a boolean function, checking if there are still replies pending on a certain command. This may not always be obvious, as the number of replies on a command is many times not known in advance.
- **seq_msgFlush**
  Does not have a corresponding CCS message system call. It flushes either the currently available replies, or waits for all pending replies from a certain environment/process.
- **seq_msgDispatch, seq_msgDispatchBreak**
  When asynchronous messages/events are enabled, *seq_msgDispatch* will be executed automatically whenever a message is available on the queue. It will pull this message from the queue, and process it as required. The Sequencer shells can deal with all standard CCS message types, as shown below:
  a. The most frequent/typical case is of course a reply (or error-reply) message. If it is of this type, the message will be copied over to dynamically allocated memory, until a request is made for this reply. If the message is a reply to a command to which a callback-script has been associated, this script will be executed.
  b. The Sequencer shells can also receive a limited set of commands. If the message is the command *SCRIPT*, it will send an immediate acknowledge, evaluate the body of the command, and return a final reply with the result of the evaluation (if this result does not fit into a single reply, seq_msgDispatch will chop it up automatically into multiple replies). If the message is any other command (except of course the commands supported by CSS, like *PING)*, it will send an error reply.
  c. Similarly, if the message is a database event, the callback-script associated to it will be evaluated. If this evaluation causes problems, the event will be disabled, so errors do not pop up continuously.
  d. Finally, if the message is of the obituary type, the script contained in the global Tcl variable *seq_obiScript* will be executed. Remark that any process interested in receiving obituaries needs to have an entry in *RtapEnvTable*, with the *Care-about-Terminations* field properly set; on the other hand, all processes doing a *ccsInit*/*ccsExit* will generate obituaries by default, ie. even when they are not listed in *RtapEnvTable*.

seq_msgDispatchBreak is a companion command, to break out from the waiting on the message queue when it is empty. The latter is never the case when *seq_msgDispatch* has been set up via *seq_ccsAsyncInput*.

Both these commands (*seq_msgDispatch* and *seq_msgDispatchBreak*) are normally not given directly by the user. *seq_msgDispatch* gets "automatically" invoked after a message of any type comes on the queue.

2.3.3 CCS On-line Database Interface

- **seq_dbDirAddrToName**
  Convert a direct address into a symbolic address. Maps into *dbDirAddrToName(3).*
- **seq_dbGetAlias**
  To retrieve the alias-name of a point. Maps into *dbGetAlias(3).*
- **seq_dbGetAttrInfo**
  To retrieve the information on an attribute. Maps into *dbGetAttrInfo(3).*
- **seq_dbGetAttrNames**
  To get the names of the attributes of a point. Maps into *dbGetAttrNames(3).*
- **seq_dbGetFamilyNames**
  To retrieve the point names of parent and children of a point. Maps into *dbGetFamilyNames(3).*
- **seq_dbGetFieldNames**
  To get the field names of a table attribute. Maps into *dbGetFieldNames(3).*
- **seq_dbGetDirAddr**
  To convert a symbolic address into a direct one. Maps into *dbGetDirAddr(3).* The direct address returned can e.g. be used to write into an OLDB table containing sampling/plotting configuration information. At present only the *seq_dbDirAddrToName* command can use this type of address as an argument.
- **seq_dbReadSymbolic**
  To read the value of an attribute. Maps into *dbReadSymbolic(3).*
- **seq_dbWriteSymbolic**
  To write a value to an attribute. Maps into *dbWriteSymbolic(3).*
- **seq_dbListCreate, seq_dbListAdd, seq_dbListRemove, seq_dbListDestroy, seq_dbListExtract, seq_dbListPut, seq_dbListList**
  To manipulate multi-read/write lists. Most of these commands have a one-to-one mapping into a CCS equivalent.
- **seq_dbMultiRead, seq_dbMultiWrite**
  The execution of the db multi-read/write operation. Equivalent to *dbMultiRead(3)* resp. *dbMultiWrite(3).*
- **seq_dbGetCwp**
  To retrieve the pathname of the current working point. Maps into *dbGetCwp(3).*
- **seq_dbSetCwp**
  To set the current working point to a pathname. Maps into *dbSetCwp(3).*
- **seq_dbLockPoint**
  To lock a db point. Maps into *dbLockPoint(3).*
- **seq_dbUnlockPoint**
  To unlock a db point. Maps into *dbUnlockPoint(3).*

Remark that *seq_dbGetDirAddr, seq_dbDirAddrToName, seq_dbLockPoint* and *seq_dbUnlockPoint* are only available in a full CCS environment (i.e. with RTAP).

2.3.4 CCS Logging System Interface

- **seq_logData**: logs a single message (up to 255 characters) on the local host.
- **seq_logTclErrors**
  This command will send all output that normally goes to *stderr* also to the CCS error log. As

this may produce quite some overhead (depending on the amount of output), its use should be limited to debugging purposes.

- **seq_logFitsAction**, **seq_logFitsComment**, **seq_logFitsEvent**, **seq_logFitsParRecord, seq_logStringParRecord, seq_logIntParRecord, seq_logRealParRecord**
  To generate FITS ops-logs

A specific command to retrieve logs can be implemented using the Tcl/TclX file access commands and the CCS logging system filtering utilities.

2.3.5 CCS Error System Interface

- **seq_errResetStack**
  To reset the error stack (used after successful recovery from error). Maps into *errResetStack(3)*.
- **seq_errAdd**
  To add an error to the error stack. Maps into *errAdd(3)*.
- **seq_errCloseStack**
  To log the and close the error stack (no error recovery possible). Maps into *errCloseStack(3)*.
- **seq_errDisplay**
  To display the current error stack in a separate window. Related to *errDisplay(3)*.
- **seq_errGetStackSize**
  To get the number of error frames currently occupying the error stack. Maps into *errGetStackSize*.
- **seq_errGetFromStack**
  Returns a particular error frame from the stack, in the same format as *seq_errPrint*. Maps into *errGetFromStack*.
- **seq_errLog**
  To reset the stack, add one error and close the stack. Does not have a CCS counterpart
- **seq_errPrint**
  Returns all the elements of the error structure corresponding to the last CCS error on the stack at the time an *errCloseStack* command was given internally by the sequencer shell. The latter happens any time when a seq-command fails due to some CCS error. Does not have a CCS counterpart

2.3.6 CCS Scan System Interface

- **seq_scanConfig <args>**: to configure the scan system; this is a simple interface on top of the CCS scan system *scanConfig* utility, passing it the list of arguments as given in **<args>**.

2.3.7 CCS Event Handling

The CCS event handling is dealing exclusively with database events. The Sequencer shells extend this concept to replies on CCS messages which were sent before (see 2.3.2). The format of these commands is for the replies identical to the one for the database events. The distinction is internally made based on the type of handle which is passed as an argument of each of the commands described below. Hence it is clear that only the database event part has a CCS counterpart.

- **seq_evtAttach**
  To attach to a database event, or to a reply event. See also *evtAttach(3)*
- **seq_evtDetach**
  To detach from a database event, or from a reply event. See also *evtDetach(3)*
- **seq_evtSingleDisable**
  To disable a database or reply event. See also *evtSingleDisable(3)*
- **seq_evtSingleEnable**
  To enable a database or reply event. See also *evtSingleEnable(3)*
- **seq_evtList**
  To list information related to database events. Has no CCS counterpart, and is not (yet) implemented for reply events.

2.3.8 Error handling

Errors occurring during the execution of commands which are Sequencer extensions to Tcl/Tk are by default logged automatically, using the standard CCS error system calls. E.g. if *seq_msgSendCommand* is used to send a command to an inactive application, *seq_msgSendCommand* will fail and the error stack will be logged automatically, identifying the Sequencer shell as topmost layer.

Similarly, if *seq_msgSendCommand* is used to send a command to a properly running application which replies with an error message on the execution of the command, the subsequent *seq_msgRecvReply* command will of course retrieve this error stack **and** log it without the need of additional instructions (done internally via an *errCloseStack* call). This default behaviour can be modified and more control over this error stack can be obtained by setting the global Tcl variables *seq_errLogging* and/or *seq_errReplyMerging* to appropriate values - see section 2.7 and the manpages of the *seq_err** commands.

On top of this feature, and the CSS error system interface (see 2.3.5), there are two Sequencer shell commands which allow additional error manipulation and logging.

- **seq_catch**
  To evaluate a command and return its completion status. This command is similar to the Tcl core command *catch*, except that eventual errors are logged to the CCS error system. This provides a tracing facility which anyhow has to be enabled very explicitly.
- **seq_logTclErrors**: See2.3.4

2.3.9 OSLX

- **seq_oslxCmd**
  To give an interface to OSLX functionality. It is accessible at script level after giving the command *package require SeqOslx*.

The OSLX functionality is available as a dynamically loadable package (see 2.6). You need to issue the command **package require SeqOslx** to have this function available in any of the Sequencer shells. The *SeqOslx* package is also available under CCS-light.

2.3.10 Convenience procedures

The Sequencer shells include a number of convenience procedures, which have no CCS counterparts. They are

- **seq_deleteHandle**
  will delete a handle plus related resources, where the handle can be either of the command, event, script or multi-read / write list type. Only for a (full or light) CCS environment.
- **seq_relToAbsPath**
  will return the absolute pathname corresponding to a relative pathname
- **seq_findFile**
  will return the absolute pathname of a file, after it has been searched for in ./, ../, $INTROOT/ and $VLTROOT (in that order).
- **seq_waitTillActive**
  will wait until a certain process is active, i.e. until it can receive CCS messages. Only for a (full or light) CCS environment.
- **seq_waitTillDead**
  will wait until a certain process is inactive. Only for a (full or light) CCS environment.
- **seq_isoTime**
  returns the current UTC time in ISO 8601 format (i.e. as ‘`yyyy-mm-ddThh:mm:ss`’).
- **seq_fitsDate**
  returns the current UTC date in the format ‘`yyyy-mm-dd`’
- **seq_timeOfDay**
  returns the current UTC time-of-the-day in the format ‘`hh:mm:ss`’.
- **seq_isoTimeToClock**
  will convert a UTC time-string in ISO-format to the internal clock value (integer, timezone adjusted) corresponding to this UTC time.

2.3.11 Debugging help

The environment installed with the seq-module includes some debugging facilities of scripts written in the Sequencer language, at different levels:

- **tclCheck**
  This is a contribution to the Tcl archive which allows to check the syntax of the script. It is a C program which can run a basic check on the syntax of a Tcl script. Very useful to detect mismatching curly braces, brackets, etc. Using this utility does not require any modification in your script
- **seq_debug**
  The Sequencer shells contain a global Tcl variable, called *seq_debug*, which is an array with all debugable *seq_** commands as elements. Assigning a particular element the value 1 (bit 0) will log the stack level, command-name plus arguments to stderr; setting bit 1 means the return-value will get printed; bit 2 stands for the timestamp, while bit 3 is for internal, command-specific debug info. If bit 1 is set, the return value is printed as “`OK`” or “`ERR`” followed by the return-string put in between greater-than and less-than signs, e.g. ‘`OK >13.` The default value of 0 means no debugging info gets printed.
  To use this feature, some instructions have to be added to your scripts, modifying the default value of 0 for the particular command(s) you want to trace. This would typically be in an initialization procedure.
- **seq_redirect**
  This command allows to redirect output sent to any file descriptor to a disk file instead. The size of the resulting file can be kept under control. The seq_debug global Tcl-variable described

above can be used in combination with this seq_redirect command, allowing to store the debug-output into a file (whose size is kept under control).

- **dputs**, **breakpoint**, **showInfo**
  These are additional commands which can be inserted anywhere in a script where more information or access to particular variables is required. This can obviously be done at numerous places. Although these are very powerful commands, it also requires some effort to enable/disable these additional instructions in the necessary places - and it doesn't work within [incr Tcl] code..
- **seqWish**
  The interactive version of *seqWish* brings up a Tk-application (based on *TkCon*, see further down in section 2.4.1) which can communicate with other Tk-based applications. In other words, the commands you type in the *seqWish* window can be executed in the interpreter of another Tk-based application on your screen. This is useful as a debugging aid of hermetic Tk- or seqWish scripts. Without modifying or editing the source of such applications directly, you can look into its variables, change the definition of procedures, make the CCS error display pop up (via the *seq_errDisplay* command), etc.
  To use this feature, click on the "*Console*" menu, then select "*Attach Console*" followed by the selection of your Tk-application under the "*Foreign Tk Interpreters*" section.
  There are two caveats related to the use of this feature, as it relies on Tk's send command:
  a. It will not work with a pure seqSh/Tcl script (i.e. it needs Tk)
  b. *xhost*-style access must be enabled, and the list of enabled hosts must be empty. This is because the *send* command is potentially a serious security loophole, since any application that can connect to your X server can send scripts to your Tk-applications. These incoming scripts can use Tcl to read and write your files and invoke subprocesses under your name.

  The latter observation is of course true for all Tk applications, and not only during debugging. More information on securing Tk applications can be obtained from the appropriate Tcl/Tk literature, the Tcl/Tk FAQs (see also section 2.8) and manpages (*xhost*(1), *xauth*(1), *send*(n), etc.).

2.4 RUNNING THE SEQUENCER SHELLS

2.4.1 Before you start

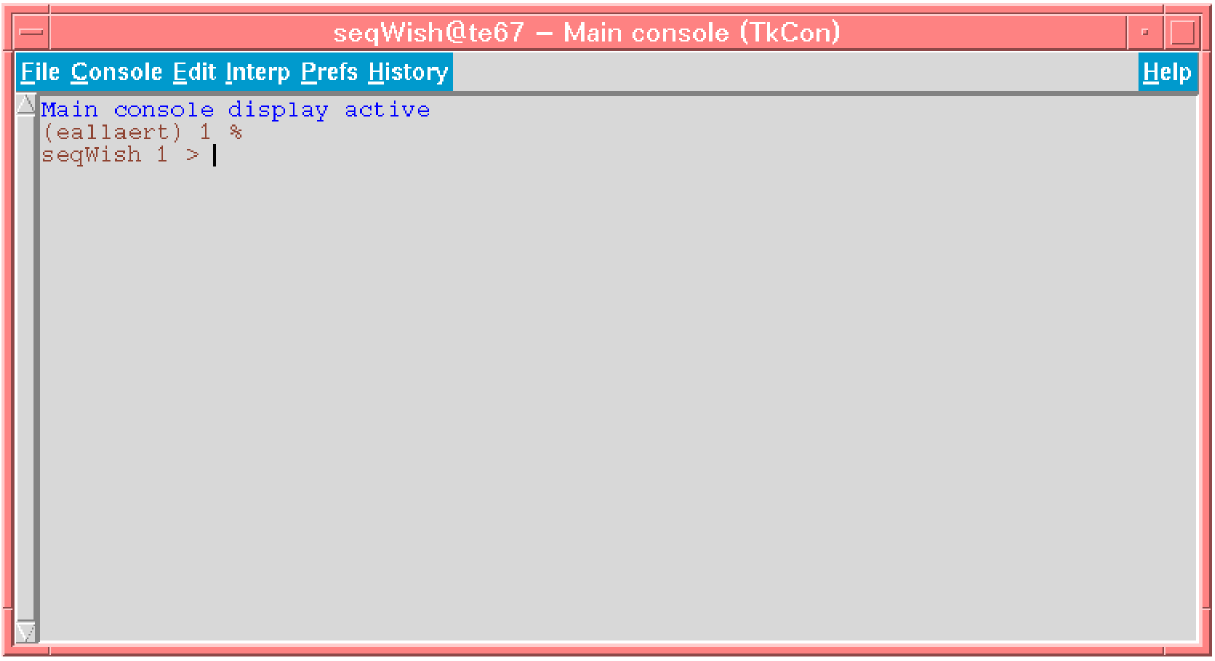
To run a Sequencer shell, ensure the binaries *seqSh* and *seqWish* are in your path and type from your login shell prompt (characters in italics are output):

```
        $ seqWish
```
[1]

A new window will pop up with the *seqWish* console (based on J. Hobbs' TkCon application), as illustrated in Figure 1. This is now the standard when *seqWish* is started up interactively. The previous behaviour can still be enforced by using the `-noTkCon` runstring option. Tkcon contains a lot of fancy stuff like command- and variable name completion, history scrolling via keyboard arrows, possibility to save stdin/stdout/stderr or the command-history to a file, etc. Extensive documentation on TkCon is available from its author via the URL `http://www.cs.uoregon.edu/research/tcl/script/tkcon/`.

The Sequencer interpreter can run interactively, as a shell, or it can also be started to execute imme-

[1] Most of the examples given here use *seqWish*; this can be substituted by *seqSh* according to your needs


seqWish@te67 – Main console (TkCon)
File Console Edit Interp Prefs History
Help

```
Main console display active
(eallaert) 1 %
seqWish 1 >
```

Figure 1: The console popping up as seqWish is started

diately a script, and finish when the script tells it has finished. The Sequencer shells can actually get their input in each of the following ways:

- Via a script file. This means the shell was invoked with the name of a script, like in

  ```
  $ seqWish ~/myScriptFile
  ```

  In this case, *~/myScriptFile* contains all Tcl/Seq commands the shell should execute. Alternatively, one can make *~/myScriptFile* executable, edit it, and put on the first line

  ```
  #!/vlt/bin/seqWish
  ```

  whereby */vlt/bin* must be replaced by the path where the seqWish binary is installed. Remark however that this line should not exceed a system defined maximum number of characters (on most systems only 32), or else *exec()* will fail - see the exec(2) manpages. This results in misleading error messages. A workaround for this can be found in most of the Sequencer scripts which are part of the VLT SW distribution, which have their header inserted by *vltMake*.
- Via the -c runstring option (only valid for seqSh, not seqWish!)

  ```
  $ seqSh -c <command>
  ```

  where <*command*> is any Tcl/Seq command to execute. It may contain a series of commands to execute, separated by `;'. The shell will execute <*command*> and then exit. This is useful to obtain on stdout some values which can be used as input for other processes. Example:

  ```
  $ seqSh -c 'seq_ccsInit; puts [seq_dbGetAlias @wte67:PARAMS]'
  ```

  will send to stdout the following string:

  ```
  <alias>PARAMS
  ```

  and then terminate. This output could be used to pipe into another process.
- Interactively, when none of the above options have been applied. In this case, a prompt will appear, inviting the user to enter the commands via the standard input device (normally the keyboard); this interactive input must be given either in the new console window or in the original terminal window from which seqWish was started, according to whether or not the -noTkCon option is specified.

```
$ seqWish -noTkCon
seqWish>_
```

To this category belongs also the typical constructs where input is piped into the shell:

```
$ echo 'seq_ccsInit; puts [seq_dbGetAlias @wte67:PARAMS]' | seqSh
<alias>PARAMS
$
```

Remark that seqWish enters the event-loop automatically when it sees an EOF; this is different from seqSh's standard behaviour, which is to terminate when it reaches the EOF. That means for the above examples with seqWish that the input should include an *exit* command, if the desired behaviour is to exit when the EOF is reached. If the other way around seqSh should not terminate, a *vwait* or *commandloop* command is needed - see the manpages of vwait(n) resp TclX(n).

Contrary to the standard Tk shell (*wish)*, you will not see a small blank square window popping up on your display if you run the shell interactively. If you want to see this window, type

```
seqWish>wm deiconify .
```

Although there are no major differences between running the Sequencer interactively vs. non-interactively, one should be aware of the following:

- In an interactive shell the result string returned by a command executed from the prompt is normally echoed back to the user. If an error occurs, the error message will be displayed preceded by the string "*Error:*". The **set** command is treated as a special case: if **set** is called with two arguments (i.e. to assign a value to a variable), the result will not be echoed. Such behaviour deviating from standard Tcl is introduced by TclX. If **set** is called with one argument (the name of the variable), then the result will be echoed, as usual.
- If the interactive shell does not recognize a command entered from the prompt, then the UNIX command path as specified in the environment variable *PATH* will be searched for a command of that name. If such command is found, it will be executed with any arguments remaining on the Tcl commandline being passed as arguments to the command. Scripts on the other hand should use the Tcl **exec** or **system** commands to run UNIX commands.
- In an interactive shell the Sequencer will most likely spend most of its time in the idle loop, after sending the prompt and while waiting for user input. During this idle loop, events (X-events, file events, CCS events,...) are handled. If the same commands are given from a script, the Sequencer shell may not go through its idle loop until the last commandline in the script is evaluated, hence it may appear not to be as responsive as when commands are given interactively. You can get around this by inserting **update** commands in your script.
- Depending on the commands given in the initialization script, an interactive windowing shell may or may not display its main window. Currently, seqWish withdraws this window at start-up, if run interactively. Give the command **wm deiconify .** if you want this window to be displayed**.**

2.4.2 Starting up

In all cases, before the Sequencer shell starts interpreting user commands, it initializes itself in the following way:

1. It initializes the Tcl, TclX, Tk, TkX, [incr Tcl], [incr Tk], Msqltcl and BLT extensions in that order for seqWish; for seqSh, it initializes Tcl, TclX, [incr Tcl] and Msqltcl. If anything fails, the Sequencer shell will tell so and quit. The list of environment variables used / needed by each

of these applications, and the list of global Tcl variables set by them can be found in the respective manpages of these extensions.

2. It initializes the Sequencer extensions:
   - it sets the version number and makes it available via the *infox appversion* command
   - it sets the global Tcl variable *seq_ccsProcName* to the string *sequencer*. This variable is used as the default process name for the seq_ccsInit command.
   - the global Tcl variable *tcl_precision* is initialized to a default value set at compilation time (see Makefile). This variable is used whenever a real is converted to a string, e.g. using the *expr* command or *seq_dbReadSymbolic*
   - it defines the global Tcl variable *seq_library*, by looking first if there is an environment variable named *SEQ_LIBRARY*; if not, it looks for the shell libraries relative to the working directory, the integration directory $INTROOT and finally the VLT root directory $VLTROOT.
   - it sets the global Tcl variable *seq_ccsType*, which reflects for what environment the Sequencer was built: *full* for CCS+RTAP, *light* for CCS-light, *none* if without CCS.
   - it defines the global Tcl variable *seq_moduleId*, and sets up the proper callbacks to ensure this is never modified to a too long or too short string. This variable is used as the module identifier in all *errAdd* calls
   - it sets the global Tcl variable *seq_ccsCmdCheck* to *NO_CHECK*
   - the Sequencer shell atomic commands are declared
   - the script file *seqInit.tcl* is evaluated. It is searched for on the path given by the global Tcl variable *seq_library* (see higher). The main task of this script is to insert *$seq_library* at the top of the list of paths searched automatically when an unknown command is given. This list is kept in the global Tcl variable *auto_path*.
     seqInit.tcl also determines your application window's icon (appearing when you press your window manager's iconify button on the application window), as follows: based on the the application name (e.g. *seqWish* or *panel*) a pixmap icon is searched for, with name <*applicationName*>*Icon.xpm*. If such colour icon exists, it will be used for the application (image name *seqIcon*, widget name *.seqIcon*). If not, a black and white bitmap with name <*applicationName*>*Icon.xbm* is searched for, which will end up as the application's icon if found. If none of the two icons are found, no icon will be defined, and the default window manager's icon will be used. This leaves the path open for easy definition of custom icons within the application's initialization code. Remark that overloading an explicitly given icon with another one may require to give the instructions

     ```
     wm withdraw .;   # unmap the window from the screen
     wm deiconify .;  # or anything else which forces a re-mapping
     ```

     in order for the window manager to notify the change. The title for the icon window is anyway set by seqWish to <*applicationName*>@<*hostName*>.

   If during any of these activities an error occurs, the Sequencer shell will inform you and quit.

3. If the Sequencer shell is run interactively, a file named *.seqShRc* resp *.seqWishRc* in the user's home directory will be sourced if it exists. This is intended as a facility to load development scripts, not as a means to support final applications, which should not rely like this on the user's environment.

From that point on, if there were no errors, user input is taken and evaluated.

If CCS functionality is required, i.e. the Sequencer shell is not just used as a shell that incorporates the TclX, BLT, Msqltcl and [incr Tcl] extensions, one should not forget to give a *seq_ccsInit* command. Try always to supply the first argument (CCS process name), or alternatively redefine the global Tcl variable *seq_ccsProcName*. Otherwise in a multiple user, multiple process environment one can end up with numerous processes which for CCS are all called Sequencer, and all commands addressed to Sequencer will be going to a single process (the one which did the last *seq_ccsInit*). Fortunately for the Sequencer shells this is only a problem for incoming commands, not replies.

2.4.3 Example Sequencer script[1]

```
#************************************************************************
# E.S.O. - VLT project
#
# seqExample.tcl
#
#
# who       when      what
# --------  --------  ----------------------------------------------
# eallaert  15/10/94  initial version
#

#***********************************************************************
#   NAME
#   seqExample - Sequencer-script for demonstration purposes
#
#   SYNOPSIS
#   seqExample
#
#   DESCRIPTION
#   This script does a number of things:
#   1. It sets the precision for reals to the maximum value
#   2. It initializes itself with CCS
#   3. It waits until another CCS application (PARTNER) is up and running
#   4. It goes into an interactive commandloop
#   5. It sends an INIT command to PARTNER and waits for a reply
#   6. 3 more commands are sent to PARTNER, without waiting for the reply
#   7. Wait for replies on the 2nd and 3rd command
#   8. Wait for replies on the 1st command
#
#   CAUTIONS
#   This script requires a PARTNER to execute!!
#
#-----------------------------------------------------------------------

# 1. We want full precision for DB double values, so let's set it up here
set tcl_precision 17

# 2. Let's declare our Sequencer to CCS, giving him a personalized name.
```

[1] The lines of code needed to make this script executable are automatically inserted by *vltMake*, provided the Makefile is properly defined, i.e. according to the VLT standards.

```
#    We’ll also want to be wide open for asynchronous commands.
seq_ccsInit seq_$env(USER)
seq_ccsAsyncInput open

# 3. We are supposed to wait here until our PARTNER is ready.  This guy
#    is started by somebody else, and he doesn’t know our name.  That
#    means we’ll have to poll him (with a PING command).  Fortunately we
#    know he’s on our environment.
while {[catch “seq_msgSendCommand $seq_ccsEnvName PARTNER PING” cmdId]} {
    sleep 1
}
seq_msgRecvReply $cmdId errNr lastReply

# 4. OK, he’s alive.  Let’s tell it and go in an interactive commandloop.
echo {
PARTNER alive and kicking....
You can now send him a few commands, or have any set of Sequencer commands
executed interactively; type CTRL-D on a fresh line to go on
}
commandloop {return $seq_ccsProcName>} {return $seq_ccsProcName=>}

# 5. Fine.  Now ready for the real stuff: let’s send PARTNER some commands
#    First comes the initialization command, and we’d better wait for a
#    positive reply on that one.
set cmdId [seq_msgSendCommand $seq_ccsEnvName PARTNER INIT]
set msg [seq_msgRecvReply -all $cmdId errNr]
if {$errNr != 0} {
    puts stderr “Something went wrong: received error msg \”$msg\””
    puts stderr “I cannot continue.  Quitting...”
    exit 1
} else {
    # bring the good news...
    echo $msg
}

# 6. Next some more commands, of which some can take their time to finish;
#    let’s send them out all, and wait for the slow ones at the end
set cmdId1 [seq_msgSendCommand $seq_ccsEnvName PARTNER CMD1 arg1]
set cmdId2 [seq_msgSendCommand $seq_ccsEnvName PARTNER CMD2 arg2 arg3]
set cmdId3 [seq_msgSendCommand $seq_ccsEnvName PARTNER CMD3]
set reply1 ““

# Some more action could come e.g. right here:
# ........

# 7. Get replies on CMD2 and CMD3 (they should finish relatively quickly).
set reply2 [seq_msgRecvReply -all $cmdId2 errNr]
if {$errNr != 0} {
    # do handling for error reply
    #....
}
set reply3 [seq_msgRecvReply -all $cmdId3 errNr]
if {$errNr != 0} {
    # do handling for error reply
    #....
}

# 8. Now we have to wait for the reply on our slow coach.
if {[catch “seq_msgRecvReply -500 $cmdId1 errNr lastReply” msg]} {
    # check if the error is simply “msg queue empty” or other
    if {![cequal $msg “reply timed out”]} {
        echo “Got an error on receiving replies on CMD1: $msg”
        echo “Cannot proceed.  Quitting...”
        exit
    }
} else {
    # seq_msgRecvReply was OK; let’s see what sort of message came in
    if {!$lastReply} {
        # more replies pending; get them all
        set reply [seq_msgRecvReply -all $cmdId1 errNr lastReply]
        if {$errNr != 0} {
            set msg $reply
        }
    }
    if {$errNr == 0} {
        # Concatenate first reply with rest of them
        set reply ${msg}\n$reply
    } else {
        # got an error reply; jump out of the procedure with an error
        error $msg
    }
}

# And so it goes on...
....
```

2.4.4 Subprocesses

The Sequencer shells can of course also schedule other processes. It has for that purpose the **exec** command. This command normally waits until its subprocess is completed, and the return value is the standard output of the child process. If on the other hand the last argument of the **exec** command is **&**,the child process will be executed in the background, and the return value will be the process id(s) of the child(ren).

If such a background process terminates, it is still possible to retrieve its exit-status within the parent process, via the **wait** command (see TclX manpage). But until such a **wait** command is issued, the exited child process will appear as a so-called *zombie* in the process table. A zombie process is a process that sent a SIGCHLD signal to its parent, while the parent did not (yet) pay attention to this signal. This misleadingly suggests something is wrong, and actually some implementations of the “top” utility even insinuate that such zombie processes still consume CPU cycles, which is not the case.

Anyway, the existence of some zombies is normally not so alarming: if such a process is launched again, the system will re-use the process table of the zombie. So if a Sequencer application exec’s only a limited set of children in the background, it is not necessary to do extra clean-up. When the parent Sequencer shell exits, its zombie-children will be removed from the process table.

If on the other hand a Sequencer application can schedule in the background an infinite set of different processes which then terminate as zombies, it could exhaust the process table if no measures are taken. There are some alternatives to cope with this (see also the TclX manpages):

- run occasionally the **wait -nohang** command, either periodically (via **after**) or whenever some background idle task is performed. Remark however that the application needs to ensure that this does not interfere with pieces of code that check if a background process has terminated (i.e. other uses of the **wait** command).
- keep track of the children process ids, and check them occasionally with the **wait** command.
- ignore the SIGCHLD signal (via the command **signal ignore SIGCHLD**). This can only be done if the application does not exec processes in the foreground, as otherwise these children would be lost: their termination would not be properly captured by the parent Sequencer shell.

2.5 THE SEQUENCER AS AN IMBEDDABLE INTERPRETER

The Sequencer comes with a C-library (**libseqC.a** and/or **libseqC.sl**) which allows it to be embedded into any application written in C-language. This opens interesting perspectives, as it means that a certain application can be partially written in C, and partially as a Sequencer script. Remark that this is quite different from e.g. sending a command to the Sequencer shell asking it to execute a certain script - in the latter case 2 processes are involved, in the former only one.

The library function which has to be called in this case to initialize the Sequencer-embedded application is **seqInitInterp**. Once this is done, the evaluation of any script can be requested calling **seqEval**.

A word of warning though: if you ask the interpreter to do a ccsInit() (passing him a script which contains a **seq_ccsInit** command), it means the interpreter will watch out for CCS messages/events in its event loop, i.e whenever a script contains **update**, or a call is made at C-level to **Tcl_MainLoop()** or **Tcl_DoOneEvent()**, and you may end up completely confused if you have your own event loop declared for such messages. In other words, one should decide whether the interpreter or the additional C code should take care of CCS messages/events, and not mix the two up.

2.6 DYNAMICALLY LOADABLE EXTENSIONS

The Sequencer shells, being based on Tcl 8.3.3, include the possibility to load binary extensions (or *packages*) on the fly. This means that in many cases you can start off with a very slim core, occupying little memory, and add packages as you are running the interpreter, without having to re-compile this core in order to make the new commands available. In other words, seqSh and seqWish can be extended with commonly available or your own packages, during runtime.

Of course such packages should comply with certain rules, in particular how they are compiled/built and how they initialize. Although several alternatives may exist - and are described in the Tcl manpages - in what follows these alternatives are narrowed down in a cookbook recipe fashion, for a loadable extension with package-name *Ext* (and VLT module name *ext*).

2.6.1 Initialization of a loadable extension

Whenever a package is loaded dynamically, it needs to get initialized, a.o. to add new commands to the interpreter. The Tcl core relies on a set of naming rules for the initialization procedure of any package, so it knows how to invoke this procedure.

The name of the initialization procedure which will be invoked is determined by the package-name. It will have the form **Ext_Init**, where *Ext* is the same as the package-name (with the first letter converted to upper case and all other letters converted to lower case). The initialization procedure must match the following prototype:

```
typedef int Ext_Init (Tcl_Interp *interp);
```

This initialization procedure must contain a call to **Tcl_PkgProvide**, like:

```
Tcl_PkgProvide (interp, “Ext”, “1.0”);
```

where the second argument is a string with the package name, and the third argument is a string with the version number, in the format <major_nr>.<minor_nr>.

At this point the initialization can proceed as usual, setting variables and creating commands with the **Tcl_CreateCommand** procedure.

The initialization procedure must return TCL_OK or TCL_ERROR to indicate whether or not it completed successfully; in the event of an error it should set interp->result to point to an error message. The result of the *load* command will be the result returned by the initialization procedure (see the manpage of the Tcl load command).

2.6.2 Compiling loadable extensions

Loadable extensions need to be created as shared libraries. This requires that the compiler generates position independent code (option *-fPIC* for gcc), and that the linker is informed that the library to build is a shared one (option *-shared* for gcc).

In general, libraries are not self-contained: they depend on a set of other libraries, like the math library (libm). Your shared library needs to be built declaring this list of dependent libraries, otherwise there may be undefined externals when seqSh/seqWish try to load this new package. Of course, these libraries need not be specified if they are already known by seqSh/seqWish, but it is always safer to declare them explicitly, just in case you’d like to load your package in another interpreter. The list of dependent libraries of a shared library or executable can be retrieved with the *chatr* utility on HP-UX or *ldd* on Solaris.

Remark that the exact path where your dependent libraries are searched at link-time and run-time depends a.o. on the setting of SHLIB_PATH under HP-UX and LD_LIBRARY_PATH on Solaris. It is highly recommended to go through the documentation of vltMake and shared libraries on your system.

2.6.3 Loading a package

Once this package has been built successfully following the rules explained above, and the resulting shared library libseq.sl has been installed in e.g. $TCLTK_ROOT/lib, it can be loaded into seqSh or seqWish, using the **load** command:

```
seqSh>load $TCLTK_ROOT/lib/libext.sl
```

If as in the example the package-name is not explicitly provided as an argument to the **load** command, Tcl will try to guess it as follows: take the last element of the library filename, strip off the first three characters (*lib*), and use any following alphabetic and underline characters as the module name, up to when a non-alphabetic character appears. So the result in our case is *ext*. As the first character is for the package handling anyway upshifted, *ext* is in this context equivalent to *Ext*, which is just fine. The resulting "guessed" name for the initialization procedure is **Ext_Init**, which is correct, and which is (or should be) part of libext.sl. **Ext_Init** will be called as soon as the actual loading of the shared library into shared memory is ready.

There are a few alternatives to this scheme, in particular via the **package** command (see the proper manpages), but they all rely one way or the other on the **load** command.

2.6.4 Examples from the Sequencer

Actually, the Sequencer is also built as a package, contained in *libseq.sl*. However, due to the RTAP licensing scheme, this package cannot be loaded dynamically, as it will result in an undefined external. When the Sequencer is built in a CCS-light environment, or without CCS, it will not contain any commercial products with restrictive license schemes like RTAP; so a shared seq-library built under CCS-light or without CCS can be loaded dynamically into any Tcl shell.

Another component of seqWish, namely the package to load pixmap images, is also built as a loadable extension, and can be used by any shell which includes Tk. The package name is *SeqXpm*, the shared library is *libseqXpm.sl*.

Finally, the *SeqOslx* extension is also built as a loadable extension. It is actually used as an example in the VLT software context, under the *example* subdirectory of the *seq* module. Check a.o. *Makefile.seqOslx*.

2.7 SEQUENCER VARIABLES

All global variables set/and used by Tcl/Tk, TclX, BLT, Msqltcl and iTcl are of course a subset of the global variables accessed by the Sequencer shells. The most important ones are:

- **argc**: number of commandline arguments
- **argv0**: the name of the executable script file or the shell's name
- **argv**: the commandline arguments; take into account that argument name-value pairs which are recognized by the shell itself should come first on the commandline, and will not be part of **argv**. E.g.
  ```
  $ seqWish -display myXTerm:0.0 -myArg1 Val1 -myArg2 Val2
  seqWish> set argv
  -myArg1 Val1 -myArg2 Val2
  ...
  ```
- **auto_path**: list of paths to search to locate Tcl scripts. Used by the *auto_load* command and the *unknown* command handler. Initialized by TclX (*TclInit.tcl*), and extended by *seqInit.tcl*
- **env**: array variable whose elements are all of the process's environment variables.
- **tcl_interactive**: set to 1 by TclX if the shell is invoked interactively
- **tcl_precision**[1]: the number of significant digits to retain when real values are converted to strings.
- **tcl_prompt1**: contains Tcl code to generate the prompt used when interactively prompting for commands. The code in this hook will be evaluated and the result will be used for the prompt. Initially set to the programs's name followed by a greater-than sign (e.g. *seqWish* >).

[1] The use of this variable is extended by the Sequencer interpreter to DB-reads as well. The Sequencer shells will also initialize this variable to a compiled-in default.

- **tcl_prompt2**: contains Tcl code to generate the prompt used when interactively prompting for continuation of an incomplete command. The code in this hook will be evaluated and the result will be used for the prompt. Initially set to =>.
- **tk_library**: set by Tk to hold the path name of the directory containing a library of standard Tk scripts and demos. This variable is set from the environment variable *TK_LIBRARY*, if it exists, or from a compiled-in default otherwise
- **tk_strictMotif**: if set to 1 by the application, Tk will try to observe Motif compliance.

The public, global variables added to this list by the Sequencer shells are:

- **seq_ccsCmdCheck**: enables/disables the CCS message system command/reply checking based on CDTs. Must be set to either *NO_CHECK* or *CHECK_CMD*. The default is *NO_CHECK*. Remark that the value of this variable at the time a command is sent has also an impact on the behaviour of *seq_msgRecvReply* for replies on this command, i.e. whether the CDT will be accessed to format the reply or not. See the *seq_msgRecvReply* manpage
- **seq_ccsEnvName**: a read-only variable, containing the name of the environment under which the Sequencer shell is running. The initial value is an empty string, and it is set by a successful **seq_ccsInit**
- **seq_ccsProcName**: initialized to *seqNAME* as defined in the include file *seq.h*. Can also be retrieved with the *infox appName* command. It is used as the default name under which to register the Sequencer shell with *seq_ccsInit*
- **seq_ccsStatus**: reflects the CCS status of the Sequencer. This variable is either undefined (no *seq_ccsInit* given), or it is set to *OPEN* or *CLOSED*, depending on whether a *seq_ccsInit* was followed by a *seq_ccsAsyncInput open* or not.
- **seq_ccsType**: a global Tcl variable, read-only, which reflects how the shell was linked. It is set to either *full*, *light* or *none*, depending on whether the shell was linked with full CCS, CCS-light or without CCS.
- **seq_debug**: a global Tcl array-variable, which reflects the debug-info flag for each of the primary seq_* commands it can handle. Each of the elements of this array can be set to 0 (default - no debug info) or have any combination of the following bits set: bit 0 (print stack level, command name plus arguments to stderr), 1 (return-value gets printed), 2 (print timestamp) and 3 (print internal, command-specific debug info). If bit 1 is set, the return value is printed as “OK” or “ERR” followed by the return-string put in between greater-than and less-than signs, e.g. ‘OK >13.1<‘. This provides basic functionality to debug scripts at the seq-level - see also section 2.3.11.
- **seq_errLogging**: a global Tcl variable reflecting how the logging of errors occuring during the execution of *seq_** commands are dealt with; *auto* (default) leads to automatic logging of all unrecoverable errors (by issuing an *errCloseStack*), *off* will not log any error (by issuing an *errResetStack*), while *manual* will not issue any *err*Stack* command nor access the “display stack”. In the latter case it is assumed that the script will take care of the error stack via “*seq_err* -internal|-reply*” commands. In all cases, the corresponding error stack will be copied over to the “display stack”.
- **seq_errReplyMerging**: a global Tcl variable which determines how the merging of error-replies will take place; *internal* will append the reply-stack to the internal error stack, *local* will make the reply-stack append to the default stack of *seq_errAdd* etc, while *off* (default) leaves the error reply stack alone. In the former two cases, the error reply stack will be reset after merging, without any logging. In the latter case, it will get logged automatically if *seq_errLogging* is set to *auto*, in which case it will also be copied over to the “display stack”.

- **seq_library**: set by the Sequencer shell to hold the path name of the directory containing a library of standard Sequencer scripts. This variable is set from the environment variable *SEQ_LIBRARY*, if it exists, or after locating the shell library scripts under either ../lib, $INTROOT/lib or $VLTROOT/lib.
- **seq_moduleId**: initialized to "*seq*", and used throughout the Sequencer interpreter for CCS error and logging system calls. The package name is also derived from this default value (presently *seq*): it is the same, with the first character converted to uppercase (i.e. *Seq*).
- **seq_obiScript**: a global Tcl variable containing the script to be executed when an obituary message is received. This script can contain some variables (single characters preceded by a %-sign) giving information about the obituary itself; they are substituted just before evaluation of the script.

2.8 INTERNET RESOURCES ON TCL/TK

There is of course a lot more information on Tcl/Tk available "out there" than ever can be contained in this User Manual, and the list is growing at an accelerating pace. As a starter, this section shows some of the Tcl/Tk documentation resources available on the Internet. These resources are given here as URLs[1].

- the new home of the Tcl/Tk core is ActiveState:

  `http://www.activestate.com`
- the central archive with most contributions/extensions to Tcl/Tk used to be at

  `http://www.NeoSoft.com/tcl/`
- recently the Tcl/Tk community acquired the tcl.tk domain:

  `http://www.tcl.tk`

  contains a wealth of links to all sorts of Tcl/Tk documentation.
- a real good place to start when one has particular questions or when looking for an exhaustive list of links or a bibliography is the FAQs at

  `http://resource.tcl.tk/resource/doc/faq/`
- there is also an on-line tutor available at

  `http://www.msen.com/~clif/TclTutor.html`
- or you can look at some Tcl programming ideoms at

  `http://www.doc.ic.ac.uk/~np2/patterns/tcl/index.html`
- the home pages for iTcl, iWidgets and BLT all used to start off from the same site:

  `http://www.tcltk.com/`

  however, this site has been down lately, and it is not clear to which other site this info is going to be relocated. In any case, the code sources for these extensions are currently on SourceForge.
- there are some html-pages ordered as an index to a set of Tcl/Tk references available on the WWW:

  `http://www.sco.com/Technology/tcl/Tcl.html`
- of course there are also newsgroups dedicated to Tcl/Tk:

  `news:comp.lang.tcl`

[1] All of these URLs were checked at the time of inclusion in this manual; of course there is no guarantee that at the time of reading they are all still correct, in particular the ones referring to home-directories of individuals.

```
news:comp.lang.tcl.announce
```

The latter newsgroup is moderated. An archive of it can be found at

```
http://www.findmail.com/list/tcl_announce/
```

3 INSTALLATION

This chapter is intended for the people who have to write / maintain the VLT Common Software installation procedures. It contains the information needed to create the proper scripts for the generation and installation of the Tcl / Tk core and extensions as part of the VLT Common Software.

3.1 REQUIREMENTS

The Tcl-extensions which are incorporated into the Sequencer must have been installed properly before the installation of the Sequencer is attempted. The cookbook recipe for that is:

1. install mSQL as documented elsewhere.
2. install the tar file for Tcl, Tk, TclX, BLT, Msqltcl, Img, TkTable, Snack and [incr Tcl] on your directory. Remark that several of these packages as provided with the VLT software distribution have been patched for known problems, i.e. one should not use tar-files downloaded straight from the internet.
3. extract each this tar-file with

   ```
   $ tar -xvof <file>.tar
   ```

   or if the tar-file was compressed, with

   ```
   $ gunzip <file>.tar.gz -c | tar -xvof -
   ```

   This will create subdirectories as appropriate
4. Make sure you will be compiling with gcc:

   ```
   $ setenv CC gcc
   ```

5. install the static version of Tcl, Tk, and [incr Tcl] by typing from your shell the following:

   ```
   $ ./configure --prefix=$TCLTK_ROOT --enable-shared=NO
   ```

   and then:

   ```
   $ make; make -k install
   ```

6. Install the shared-lib version of Tcl, Tk, and [incr Tcl] by typing from your shell the following:

   ```
   $ rm -rf config.cache ; make clean
   $ ./configure --prefix=$TCLTK_ROOT --enable-shared
   ```

   and then:

   ```
   $ make; make -k install
   ```

7. Install TclX as follows (it does the static and shared-lib versions in one go)

   ```
   $ cd tclX8.3/unix
   $ ./configure --prefix=$TCLTK_ROOT --enable-shared --with-help
   $ make; make -k install
   ```

8. Install BLT as follows:

   ```
   $ cd blt2.4u
   $ ./configure --prefix=$TCLTK_ROOT --enable-shared
   ```

   and then:

   ```
   $ make; make -k install
   ```

9. Install MsqlTcl as follows:

   ```
   $ cd ../msqltcl-1.99
   ```

```
$ ./configure --prefix=$TCLTK_ROOT --enable-shared
 --with-msql-include=$TCLTK_ROOT/include
 --with-msql-library=$TCLTK_ROOT/lib
```

and then:

```
$ make; make -k install
```

10. Install the Img extension as follows:

```
$ cd ../img1.2.4
$ ./configure --prefix=$TCLTK_ROOT --enable-shared --disable-
stubs
```

and then:

```
$ make; make -k install
```

11. Install TkTable as follows:

```
$ cd ../Tktable2.7/unix
$ ./configure --prefix=$TCLTK_ROOT --enable-shared
```

and then:

```
$ make; make -k install
```

12. Install Snack as follows:

```
$ cd ../snack2.1/unix
$ ./configure --prefix=$TCLTK_ROOT --enable-shared \
 --with-tcl=../../tcl8.3.3/unix --with-tk=../../tk8.3.3/unix \
 --disable-stubs
```

and then:

```
$ make; make -k install
```

This will leave all the required libraries, programs and scripts in directories under the *$TCLTK_ROOT* hierarchy, e.g. *$TCLTK_ROOT/lib* and *$TCLTK_ROOT/lib/tclX8.3*, hence you will need write access to these. If you do not (want to) have the *$TCLTK_ROOT* directory on your system, scan through the *README* and *INSTALL* files of all these packages for precise instructions.

3.2 COMPILATION AND INSTALLATION OF THE SEQUENCER

The standard procedures to install VLT SW modules apply to the Sequencer:

1. Set your working directory to the VLT SW installation directory:

```
cd <VLTSW>/INSTALL
```

2. Compile and install the Sequencer (as part of HOS):

```
./buildHOS
```

3. Make sure you have a clean environment; if necessary, reboot your workstation.

If the environment variable *NOCCS* exists when the installation procedure is started, the Sequencer shells will not contain any CCS extensions, i.e. they will be Tcl/Tk (windowing) shells containing the extensions as described in the introduction of 2.3. Otherwise, if *RTAPROOT* exists, the CCS functionality based on RTAP calls will be included. If both *NOCCS* and *RTAPROOT* are undefined, the CCS-light libraries will be used.

Remark that the settings of *NOCCS* and *RTAPROOT* should reflect the real status of the workstation the Sequencer shells are generated and supposed to be executed on. Trying to generate e.g. a CCS-light seqWish on a workstation equipped with RTAP and full CCS is not a good idea, and may fail at any one stage.

3.3 VERIFICATION

The proper installation of the Sequencer can be tested as follows:

- start up the Sequencer shell:

  ```
  $ seqWish -noTkCon
  ```

  The sequencer prompt (*seqWish>*) should show up in the window where you started the Sequencer from.

- type at the sequencer prompt:

  ```
  wm deiconify .
  ```

  Now a small window (the Tk main window) should pop up. Continue with

  ```
  info commands seq*
  ```

  and a list of all the Sequencer extensions to Tcl should be printed. Do the same thing for BLT:

  ```
  namespace eval ::blt info commands *
  ```

  and [incr Tcl]

  ```
  info command itcl*
  ```

- exit the shell:

  ```
  exit
  ```

If any of this fails, the error messages should be a first and concise help for what went wrong. One should in particular pay attention to the availability of script files on the right directories.

4 REFERENCE

tclCpp(1)

```
NAME
tclCpp - do cpp-style pre-processing on a Tcl script


SYNOPSIS
tclCpp -p [-s] <file>
tclCpp [-b] [-c] [-D name[=value]] [-I pathname] <file>


DESCRIPTION
tclCpp pre-processes <file> with cpp.  This means <file> can contain
cpp-style pre-processor directives, like #include and #define statements.
Apart from these pre-processor directives, <file> needs to be written
according to the Tcl syntax. To avoid confusion between Tcl commentlines
and pre-processor directives (both starting with a #-sign), the following
rule applies: lines starting with a hash sign followed immediately by a
lowercase alphabetic character are considered pre-processor directives;
otherwise they are considered Tcl commentlines.

In its first form (-p option), <file> is a Tcl script, which gets its
commentlines prettified/prepared for the cpp preprocessing.  I.e. lines
starting with a hash sign followed immediately by a lowercase alphabetic
character get a space inserted after the #-sign.
  -p : prettify commentlines
  -s : do not substitute /* and */ by their hexadecimal representation.
       By default tclCpp will replace these strings, in order to avoid
       conflicts with cpp, which considers them the start resp end of
       a comments block.
The output is a file named <file>.cpp

In the second form, the actual pre-processing takes place.
  -b : keep blank lines; cpp may actually insert a few blank lines.  By
       default blank lines are removed.
  -c : keep comment lines.  They are removed by default.
  -D : additional #define directives. tclCpp itself defines the MAKE_TCL
       macro
  -I : pathnames for #include directives.
  <file> : Tcl script file, extended with pre-processor directives.  Needs
       to have a .cpp extension.
The output is a file named as <file> but without the .cpp extension.


ENVIRONMENT
GCC | CC: used to locate the C preprocessor to be used. The default is
"gcc".


CAUTIONS
tclCpp -p will replace the /* and */ strings by default, even in comment
lines and within curly braces, by \x2f* resp *\x2f (\x2f being the
hexadecimal notation for "/").  If this is not desired, specify also the
-s option,  and ensure that these strings will not disturb cpp.


RETURN VALUES
0 if everything OK
1 in case of errors

tclCpp also creates an output file as described above.
```

```
EXAMPLES
.....
$ tclppp -p myScript.tcl
$ ... (edit myScript.tcl.cpp to add pre-processor directives)
$ tclppp myScript.tcl.cpp
.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
cpp(1)




- - - - - -
Last change:  28/03/02-10:36

seqEval
```

See seqInitInterp(3).

- - - - - -

Last change: 28/03/02-10:36

seqInitInterp(3)

```
NAME
seqInitInterp, seqEval - create and initialize a Sequencer interpreter
            and evaluate a Sequencer script


SYNOPSIS
#include <seq.h>

ccsCOMPL_STAT seqInitInterp (char           *reserved1,
                             char           *reserved2,
                             vltLOGICAL     interactive,
                             Tcl_Interp     **interp,
                             ccsERROR       *error)

ccsCOMPL_STAT seqEval (Tcl_Interp       *interp,
                       char             *script,
                       ccsERROR         *error)


DESCRIPTION
seqInitInterp creates and initializes a Sequencer interpreter.

  reservedN : dummy string
  interactive: flag to signal if this is an interactive application (like a
               shell), i.e. if typed commands have to be expected.
               If true, the Tcl-variable "tcl_interactive" is set to 1.
               This is needed for the initialization of TclX
  interp   : pointer to the returned interpreter-pointer
  error    : CCS error structure pointer

  seqInitInterp will call the Tcl_AppInit() function to initialize the
  interpreter. The libseq library contains such a function, which is
  identical to the Tcl_AppInit() function used by seqSh. In other words,
  the version of Tcl_AppInit included in libseq initializes all extensions
  just as seqSh does, i.e. [incr Tcl], Tcl, TclX and seq.  If more
  extensions are needed, like e.g. SeqOslx, they have to be loaded
  dynamically via seqEval (see the Tcl "load" and "package" commands).

  Remark that if Tcl_AppInit() includes the initialization of Tk, the Tk
  main window can only appear or get updated when the Sequencer event-loop
  is entered, i.e. by calling from your application either Tcl_Mainloop()
  once or Tcl_DoOneEvent() repeatedly.  In other words, if your application
  does not include such calls, no window will appear.

seqEval evaluates a script in the context of the interpreter it is passed
  interp   : initialized interpreter (cf seqInitInterp)
  script   : string containing any valid script to execute (e.g. "source
             myScriptFile" or "set a [seq_dbReadSymbolic <alias>myScalar]")
  error    : CCS error structure, filled out only if the evaluation fails


CAUTIONS
The application calling this function is responsible for including the
proper libraries, i.e. a library containing the Tcl_AppInit function.
Remark that the libseq library includes such a function, which is identical
to the one used for seqSh, i.e. it does not include any initialization of
Tk and related extension. The seq/src/tkXAppInit.c source can be used as
an example Tcl_AppInit function including this Tk initialization.


SEE ALSO
```

```
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X



- - - - - -
Last change:  28/03/02-10:36
```

Seq_Init(3)

```
NAME
Seq_Init - initialization procedure for the Sequencer specific commands


SYNOPSIS
#include "seqPrivate.h"

int Seq_Init(Tcl_Interp *interp)


DESCRIPTION
This function initializes the interpreter for the commands that are
specific for the Sequencer.  This comes down to initializing a few
global Tcl variables, declaring the new commands, and finally the
execution of the initialization script seqInit.tcl


FILES
seqInit.tcl is evaluated at the end, i.e. after all commands have been
successfully declared.


ENVIRONMENT
The environment variable SEQ_LIBRARY gives the pathname where a number of
predefined Sequencer scripts are looked for, including the initialization
script seqInit.tcl.

seq_library is a global Tcl variable, derived from the environment variable
SEQ_LIBRARY.  If this environment variable is not set, seq_library will be
set to one of the following paths, in this order of precedence:
  - ../lib/libseqTcl.tcl
  - $INTROOT/lib/libseqTcl.tcl
  - $VLTROOT/lib/libseqTcl.tcl

The environment variable SEQ_DEFAULT_PREC gives the default precision that
will be used for printing floats, and for conversion of database values to
strings.

tcl_precision is a global Tcl variable which is writable, and which is set
to SEQ_DEFAULT_PREC. If this environment variable is not set, a default
precision which was given at compilation time will be used (see Makefile).
tcl_precision is also linked to the global C-variable seqPrecision.

seq_ccsProcName is a global Tcl variable which is read-only, initialized to
seqNAME as defined in seq.h, and linked to the global C-variable
myProcName.  This C-variable is used as the default name under which to
register the Sequencer with CCS - see seq_ccsInit.

seq_moduleId is a global Tcl variable which is writable, initialized to
"seq", and linked to the global C-variable seqModuleId of type ccsMODULEID.
This C-variable is used only in the seq_errAdd command, and allows to
modify the moduleId-argument of the underlying call to errAdd(3).

seq_errLogging is a global Tcl variable which is writable, initialized to
"auto".  It reflects how the logging of errors occuring during the
execution of seq_* commands should be dealt with; "auto" leads to automatic
logging of all unrecoverable errors (by issuing an errCloseStack()), "off"
will not log any error (by issuing an errResetStack), while "manual" will
not issue any err*Stack command, assuming that the script will take care
of this (see seq_err* commands).  In all cases the error stack remains
accessible with the seq_errGetStackSize and seq_errGetFromStack commands.
```

seq_errReplyMerging is a global Tcl variable which is writable, initialized to "off". It reflects how the error stacks received as part of message replies are dealt with; "off" means an error reply will not be merged into any other error stack, "internal" means it will end up at the bottom of the so-called internal error stack, while "local" means it will get appended to the default error stack accessed by seq_errAdd(n) and company (see also seq_errResetStack(n)).

seq_ccsCmdCheck is a global Tcl variable which is writable, initialized to "NO_CHECK", and linked to the global C-variable seqCmdCheck of type msgCHECKFLAG. This C-variable is in seq_msgSendCommand passed on to msgSendCommand(3). Allowed values for seq_ccsCmdCheck are NO_CHECK and CHECK_CMD. It is also used in seq_msgRecvReply, to decide whether formatting of replies should be attempted by default (i.e. it won't if seq_ccsCmdCheck is set to "NO_CHECK").

seq_ccsType is a global Tcl variable, read-only, which reflects how the shell was linked. It is set to either "full", "light" or "none", depending on whether the shell was linked with full CCS, CCS-light or without CCS.

seq_debug is a global Tcl array-variable, which reflects the debug-info flag for each of the primary seq_* commands it can handle. Each of the elements of this array can be set to 0 (default - no debug info) or have any combination of the following bits set: bit 0 (print stack level, command name plus arguments to stderr), 1 (return-value gets printed), 2 (print timestamp) and 3 (print internal, command-specific debug info).
If bit 1 is set, the return value is printed as "OK" or "ERR" followed by the return-string put in between greater-than and less-than signs, e.g. 'OK >13.1<'.
This provides basic functionality to debug scripts at the seq-level.

RETURN VALUES

TCL_OK if no errors occurred.
TCL_ERROR if one of the following conditions is met (with the corresponding error message between quotes):

- error message left by Tcl_LinkVar if tcl_precision could not be linked to seqPrecision
- error message produced by evaluating seqInit.tcl, which contains an error.

The callback routine which checks the modification of seq_moduleId will produce an error whenever the new value contains less than 3 or more than 6 characters.

CAUTIONS

Renaming a seq_* command will not affect this command's name in the seq_debug array. So after renaming, the original name has still to be used to modify the debug-flag.
Some debugable seq_* commands are implemented as Tcl procedures (seq_catch, seq_errDisplay and seq_msgFlush). Each of these procedures calls itself one or more seq_* commands implemented in C. If all of their debugging flags are on, you will see a "nesting" of debugging output.

Remark that due to the way CCS keeps track of the environment name, changing the RTAPENV environment variable *after* any CCS-function has been called (explicitly via a seq_* command, or implicitly as the result of calling a seq_* command, e.g. when errors are logged) will have no effect on CCS.
Example:

```
   seqSh> seq_dbSetCwp :         ; # fails if no seq_ccsInit given before
   Error: cannot set CWP to ":"
   seqSh> set env(RTAPENV)
   wte5
   seqSh> set env(RTAPENV) wtetest
   seqSh> seq_ccsInit me         ; # will do a ccsInit in wte5, not wtetest
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
tkXAppInit(3), tclXAppInit(3)

- - - - - -
Last change: 28/03/02-10:36

breakpoint(n)

```
See  dputs(n).


- - - - - -
Last change:  28/03/02-10:36
```

dputs(n)

```
NAME
dputs, breakpoint, showInfo - A couple of Tcl-debug procedures


SYNOPSIS
source seqDebug.tcl

dputs ?<args>?
breakpoint ?<level>?
showInfo ?<level>?


DESCRIPTION
dputs: prints out the <args> to stderr, but only if the global Tcl variable
       "Debug" is set to the name of the procedure we want to debug, or to
       "toplevel" if the toplevel is to be debugged. If no arguments are
       passed, a call to this procedure is equivalent to setting a
       breakpoint (provided "Debug" is set to the name of the procedure to
       debug).

breakpoint: set a breakpoint.  Four commands are interpreted directly by
            the breakpoint procedure itself:
               + to move up the call stack
               - to move down the call stack
               ? to print information about the current stack frame
               C return from a breakpoint
            All other commands are evaluated in the current stack frame
            The optional argument <level> indicates how many more levels to
            go up, in case several of these debug-procedures are nested
            (e.g. when dputs calls breakpoint)

showInfo: print useful information about the stack level (stack level,
          procedure name and arguments) to stderr.

Based on: "PracTcl Programming Tips", Linux Journal, October 1995
by Stephen Uhler


RETURN VALUES
as any Tcl procedure


CAUTION
The breakpoint proc seems to have some problems with namespaces, i.e
when in a child namespace it can complain that perfectly fine procedures
are not procedures. It looks like a bug in the "uplevel" command.



- - - - - -
Last change:  28/03/02-10:36
```

seq_catch(n)

```
NAME
seq_catch - evaluate a command and return its completion status


SYNOPSIS
seq_catch <command> ?<msg>? ?<sourceId>?


DESCRIPTION
seq_catch has an identical syntax and behaves as the Tcl "catch" command.

<command>  : any Sequencer script
<msg>      : variable name where to store the result string for <command>
<sourceid> : string containing location identifier; remark that only the
             first word of this string will be transmitted to the CCS
             error logging system

On top of what "catch" does, seq_catch logs the error (if any) produced
by <command> to the CCS error logging system.  For that purpose it can use
the optional <sourceId> argument, which is for the error system known as
the "location identifier".  Typically the caller could set <sourceId> to
[info script].  If <sourceId> is not specified, it will be set to the name
of the script file which executes this seq_catch command.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
identical to "catch":

returns OK, with the result string set to "0" if <command> did not produce
an error.  If <command> returned an error, the result string contains 1
and <msg>, if given, will be set to the error message left by <command>.

returns ERROR with corresponding message in the result string under the
following conditions:
- "wrong # args: should be "seq_catch <command> ?<msg>? ?<sourceId>?"" if
  there were more than 2 arguments given to this <command>.


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>seq_catch {set a $b} msg "MyProcedure"
0
seqWish>seq_catch {set a $c} msg "MyProcedure"
1
seqWish>set msg
can't read "c": no such variable
seqWish>.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
catch(n), seq_errAdd(n)
```

```
- - - - - -
Last change:  28/03/02-10:36
```

seq_ccsAsyncInput(n)

```
See  seq_ccsInit(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_ccsExit(n)

```
See  seq_ccsInit(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_ccsInit(n)

```
NAME
seq_ccsInit, seq_ccsExit, seq_ccsAsyncInput - interface to ccsInit, ccsExit
                    and the asynchronous input of messages


SYNOPSIS
seq_ccsInit ?<procName>?

seq_ccsExit

seq_ccsAsyncInput open|close


DESCRIPTION
seq_ccsInit:
    calls ccsInit to register the Sequencer in CCS

    <procName> : string containing name under which to register the Sequencer.

    seq_ccsInit sets also the global Tcl variable "seq_ccsStatus" to
    "CLOSED".  If <procName> is not given, it defaults to the value of the Tcl
    variable seq_ccsProcName.
    If all went well, the global Tcl variable "seq_ccsEnvName" will be set
    to the name of the local environment.

seq_ccsExit:
    calls ccsExit and unsets the global variable "seq_ccsStatus".
    Whenever seq_ccsExit command completes successfully, internal lists
    containing info about commands with pending replies are cleared,
    i.e. seq_msgList will return an empty list.  If the seq_ccsExit command
    is given within the processing of a SCRIPT command (see
    seq_msgDispatch), it will inform the originator(s) of this command that
    it is exiting, sending a last reply with an error number 1.

seq_ccsAsyncInput:
    enables or disables the treatment of commands that are sent to the
    Sequencer via the CCS message system.

    open  : enable the input of commands
    close : disable the input of commands


ENVIRONMENT
RTAPENV:
   an environment variable, used to determine under which CCS-environment
   the application needs to registered by the seq_ccsInit command. See also
   the CAUTIONS section.

seq_ccsEnvName:
   a read-only global Tcl variable, initialized after the first seq_ccsInit
   command to the string-value given by CCS as the local environment name.

seq_ccsProcName:
   a read-only global Tcl variable, initialized by seqInit to seqNAME as
   defined in seq.h, and linked to the global C-variable myProcName.
   This variable is used as the default name under which to register the
   Sequencer with CCS.  If the seq_ccsInit command includes the <procName>
   parameter, seq_ccsProcName will be updated accordingly after a
   successful ccsInit.

seq_ccsStatus:
   a read-only global Tcl-variable, which reflects the status of
   initialization the Sequencer went through; values can be:
   - CLOSED : "seq_ccsInit" has been given before
   - OPEN:    "seq_ccsAsyncInput open" was given (after "seq_ccsInit"
              succeeded)
   - if the variable does not exist, "seq_ccsInit" was never given before
     or "seq_ccsExit" was given after "seq_ccsInit".


RETURN VALUES
OK if no errors occurred, accompanied by
  - for seq_ccsInit the Tcl filedescriptor used for the message queue
    monitoring
  - an empty reply for all other commands.

ERROR if one of the following conditions is met (corresponding error
message is here between quotes):
  - 'wrong # args: should be "seq_ccsInit ?<procName>?"', 'wrong # args:
    should be "seq_ccsExit"' or 'wrong # args: should be "seq_ccsAsyncInput
    open|close"' if the command was given with an improper amount of
    arguments
  - 'ccsInit failed' if ccsInit failed
  - 'ccsGetMyProcId failed' if ccsGetMyProcId failed
  - '"seq_ccsInit" was given before' if more than one seq_ccsInit command
    is given without a seq_ccsExit in between (and remark that CCS expects
    only one pair of init/exit commands per application).
  - message left by Tcl_LinkVar, if the Tcl global variable seq_ccsStatus
    or seq_ccsEnvName could not be linked to a C variable after ccsInit
    succeeded.
  - 'ccsExit failed' if ccsExit failed
  - message left by Tcl_UnsetVar2, if the global variable seq_ccsStatus
    could not be unset after ccsExit succeeded.
  - 'bad argument: "<arg>" should be open or close' for a seq_ccsAsyncInput
    command with an improper argument
  - 'use seq_ccsInit first' if seq_ccsAsyncInput is given before
    seq_ccsInit
  - 'call to seqMonitorQ failed' if seq_ccsAsyncInput got an error returned
    by this routine.
CCS errors are also logged via the CCS error logging system.


CAUTIONS
Under CCS-lite, the current setting of RTAPENV is taken whenever the first
CCS-function gets called, for CCS internal needs. If this first CCS-function
is not a ccsInit(), this value of RTAPENV will anyway be used later when
ccsInit() gets called, independent of RTAPENV's value at that later time.
This means that e.g. in the wtctest environment, issuing first a
"seq_errResetStack" command followed by a "set env(RTAPENV) xyz" and a
"seq_ccsInit" command, will lead to a ccsInit() in the wtctest
environment instead of the xyz environment.
This is not inherent to seq, but rather to CCS-lite. In fact, to enable an
effective modification of the environment variable RTAPENV within a
Sequencer script (e.g. based on runstring options), the Sequencer's internal
initialization does not call any CCS-function under CCS-lite, thereby
shifting the responsibility to do things in the proper order to the
application. Remark that there are several seq_* commands, apart from the
obvious ones, that (could) lead to calls of CCS-functions, e.g. seq_catch,
seq_errLog, seq_logTclErrors and seq_errDisplay.

Repeated seq_ccsExit commands will normally succeed as repeated ccsExit()
calls succeed. Use the Tcl-variable seq_ccsStatus if you want to know what
CCS-state the Sequencer-shell is in.
```

The handling of the seq_ccsInit command includes a rtMonitorQueue() call. Due to what seems to be a bug in the latter, after messages came on the queue this file descriptor will not be released by seq_ccsExit. In other words, seq_ccsInit/seq_ccsExit cycli may lead to high-numbered file descriptors, and even to some memory leaking. Remark that ccsInit/ccsExit cycli are not officially supported by CCS.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the 'seqWish>' prompt, and all replies on a new line)

```
....
seqWish>seq_ccsInit
file5
seqWish>echo $seq_ccsStatus
CLOSED (*)
seqWish>seq_ccsInit
Error: "seq_ccsInit" was given before
.....
seqWish>seq_ccsAsyncInput open
seqWish>echo $seq_ccsStatus
OPEN (*)
seqWish>seq_ccsExit
seqWish>echo $seq_ccsStatus
can't read "seq_ccsStatus": no such variable
seqWish>seq_ccsExit
seqWish>....
```

(*): These 3 characters are only there to avoid that docDoManPages interprets the words CLOSED and OPEN as new sections of the manpage

SEE ALSO

'Tcl and the Tk toolkit', John K. Ousterhout, ISBN 0-201-6337-X
seq_msgDispatch(n), seqCcsInit(3), seqCcsExit(3), ccsInit(3), ccsExit(3), seqInit(3)

- - - - - -
Last change: 28/03/02-10:36

seq_dbDirAddrToName(n)

```
See  seq_dbGetDirAddr(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetAlias(n)

```
See  seq_dbGetAttrNames(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetAttrInfo(n)

NAME
seq_dbGetAttrInfo - get the detailed characteristics of an attribute

SYNOPSIS
seq_dbGetAttrInfo <attrSymAddress>

DESCRIPTION
seq_dbGetAttrInfo will retrieve the detailed description of an attribute.

```
<attrSymAddress> : string containing a symbolic address of the attribute,
                   in a format as permitted by dbGetAttrInfo(3)
```

seq_dbGetAttrInfo will return 4 values, in the following order:

- the attribute type (SCALAR, VECTOR or TABLE)
- a list of data types. In the case of SCALARs and VECTORs this list contains a single element (without braces). For TABLEs this list contains one element per field; if no field indices were specified, the data types of all fields will be returned.
- the number of records; always 1 for SCALARS; for TABLES it is according to the record indices, or the total number of records in the table if no record indices were given
- the number of fields; always 1 for SCALARS and VECTORs; for TABLES it is according to the field indices, or the total number of fields in the table if no field indices were given

ENVIRONMENT
no environment nor Tcl variables are read nor set

RETURN VALUES
OK return if no errors occurred. In this case the result string contains the attribute type, the list of data types (a single element for SCALARs and VECTORs), the number of records and number of fields.
ERROR return if one of the following conditions is met (with corresponding error message here between quotes):

- "wrong # args : should be "seq_dbGetAttrInfo <attrSymAddress>": the command was given with the wrong number of arguments;
- "cannot get info on "<attrSymAddress>": dbGetAttrInfo returned an error for this argument

EXAMPLES
(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_dbGetAttrInfo <alias>eric.myScalar
SCALAR 5 1 1
seqWish>seq_dbGetAttrInfo <alias>eric.myVector
VECTOR 6 15 1
seqWish>seq_dbGetAttrInfo <alias>eric.myTable
TABLE {20 24 25} 5 3
seqWish>seq_dbGetAttrInfo <alias>eric.myTable(0:0,1:2)
TABLE {24 25} 1 2
seqWish>.....
```

```
SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbGetAttrInfo(3), dbGetDirAddr(3)




- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetAttrNames(n)

```
NAME
seq_dbGetAttrNames, seq_dbGetFamilyNames, seq_dbGetFieldNames,
seq_dbGetAlias -
     list the attributenames of a point, get the symbolic addresses of
     parent and children, list the field names of a table or get the
     alias name of a point


SYNOPSIS
seq_dbGetAttrNames <pointName>

seq_dbGetFamilyNames <pointName> <view>

seq_dbGetFieldNames <attrName>

seq_dbGetAlias <pointName>


DESCRIPTION
seq_dbGetAttrNames:
  will retrieve the list of names of the attributes belonging to a point

  <pointName> : string containing a symbolic address of the point whose
                attributenames we want to know

  seq_dbGetAttrNames will return a single list containing the names of
  the attributes.

seq_dbGetFamilyNames:
  will retrieve the symbolic addresses of the parent and the children of
  a point.

  <pointName> : string containing a symbolic address of the point whose
                family addresses we want to know

  <view>      : view specifier for the addresses; can be either ABSOLUTE,
                ALIAS or RELATIVE (both upper- and lowercase allowed)

  seq_dbGetFamilyNames will return a list containing the following values:
  1. the parent's address
  2. the addresses of the children, one list-element per child.
  If the point does not have any children, this list will contain only one
  element, i.e. the parent's address.

seq_dbGetFieldNames:
  will retrieve the list of names of the fields of a TABLE attribute

  <attrName> : string containing a full symbolic address of the attribute
               whose fieldnames we want to know

  seq_dbGetFieldNames will return a single list containing the names of
  the fields.

seq_dbGetAlias:
  will retrieve the alias name of a point

  <pointName> : string containing a symbolic address of the point whose
                alias we want to know

  seq_dbGetAlias will return the alias name.
```

ENVIRONMENT

no environment nor Tcl variables are read nor set

RETURN VALUES

```
OK return if no errors occurred.  In this case the result string contains
    the attribute, family or alias names, as indicated above.
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - "wrong # args : should be "seq_dbGetAttrNames <pointName>": the
    command was given with the wrong number of arguments
  - "wrong # args : should be "seq_dbGetFamilyNames <pointName> <view>":
    the command was given with the wrong number of arguments;
  - "wrong # args : should be "seq_dbGetFieldNames <attrName>": the
    command was given with the wrong number of arguments;
  - "wrong # args : should be "seq_dbGetAlias <pointName>": the command
    was given with the wrong number of arguments;
  - "wrong view specifier: should be ABSOLUTE, ALIAS or RELATIVE"
  - "cannot get attribute names of "<pointName>": dbGetAttrNames returned
    an error for this argument
  - "cannot get family info on "<pointName>": dbGetFamilyNames returned
    an error for this argument
  - "cannot get field names of "<pointName>": dbGetFamilyNames returned
    an error for this argument
  - "cannot get alias name of "<pointName>": dbGetAlias returned an error
    for this argument
```

CAUTIONS

Unfortunately, point and attribute names can in principle contain spaces. If this is the case, such name will be surrounded by braces, as any proper Tcl-list element.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_dbGetAttrNames :PARAMS:SCALARS
scalar_double scalar_float scalar_int32 scalar_logical scalar_string128
seqWish>seq_dbGetFamilyNames :PARAMS:SCALARS absolute
:PARAMS
seqWish>seq_dbGetFamilyNames :PARAMS absolute
: :PARAMS:SCALARS :PARAMS:TABLES :PARAMS:VECTORS
seqWish>seq_dbGetFamilyNames :PARAMS alias
<alias>root <alias>SCALARS <alias>TABLES <alias>VECTORS
seqWish>seq_dbGetAttrNames :PARAMS:TABLES
full_table
seqWish>seq_dbGetFieldNames :PARAMS:TABLES.full_table
logical int8 uint32 float string4 string32
seqWish>seq_dbGetAlias :PARAMS:TABLES
<alias>TABLES
seqWish>.....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbGetFamilyNames(3)

```
- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetCwp(n)

```
See  seq_dbSetCwp(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetDirAddr(n)

NAME

seq_dbGetDirAddr, seq_dbDirAddrToName - get the internal address of a point or attribute, or convert the direct address into a symbolic one.

SYNOPSIS

```
seq_dbGetDirAddr <attrSymAddress>

seq_dbDirAddrToName <directAddress> <
```

DESCRIPTION

```
seq_dbGetDirAddr will retrieve the detailed description of an attribute.
  <attrSymAddress> : string containing a symbolic address of the attribute,
                     in a format as permitted by dbGetDirAddr(3);
  seq_dbGetDirAddr will return a formatted string which is understood by
  several db/Rtap routines and utilities as a direct address.

seq_dbDirAddrToName will convert the direct address into a symbolic one
  <directAddress> : string with a direct address of a point/attribute;
  <view>          : view specifier for the address; can be either ABSOLUTE,
                    ALIAS or RELATIVE (both upper- and lowercase allowed)
  seq_dbDirAddrToName will return the corresponding address in the form
  as specified in <view>
```

ENVIRONMENT

no environment nor Tcl variables are read nor set

RETURN VALUES

```
OK return if no errors occurred.  In this case the result string contains
    the address.
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - "wrong # args : should be "seq_dbGetDirAddr <attrSymAddress>": the
    command was given with the wrong number of arguments;
  - "wrong # args : should be "seq_dbDirAddrToName <directAddress> <view>":
    the command was given with the wrong number of arguments;
  - "cannot get internal address of "<attrSymAddress>"": dbGetDirAddr(3)
    returned an error for this argument
  - "wrong view specifier: should be ABSOLUTE, ALIAS or RELATIVE";
  - "cannot convert "<directAddress>" to a symblic address":
    dbDirAddrToName(3) returned an error for this argument
```

CAUTIONS

Suffering from the same limitations as dbGetDirAddr(3) resp. dbDirAddrToName(3) these commands work only for the local environment (i.e. there should be no environment-name part in <attrSymAddress> nor <directAddress>)

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_dbGetDirAddr <alias>eric
@[10]
```

```
seqWish>seq_dbGetDirAddr <alias>eric.myVector
@[10.0]
seqWish>seq_dbGetDirAddr <alias>eric.myVector(1:2)
@[10.0(1:2)]
seqWish>seq_dbDirAddrToName @[10.0] alias
<alias>eric.myVector
seqWish>.....
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbGetDirAddr(3), dbGetDirAddr(3), seqDbDirAddrToName(3),
dbDirAddrToName(3)

- - - - - -
Last change: 28/03/02-10:36

seq_dbGetFamilyNames(n)

```
See  seq_dbGetAttrNames(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbGetFieldNames(n)

```
See  seq_dbGetAttrNames(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbListAdd(n)

See seq_dbListCreate(n).

- - - - - -

Last change: 28/03/02-10:36

seq_dbListCreate(n)

```
NAME
seq_dbListCreate, seq_dbListAdd, seq_dbListRemove, seq_dbListDestroy,
seq_dbListPut, seq_dbListList
      - commands for the handling of db multi-read/write lists


SYNOPSIS
seq_dbListCreate <environment> <access>

seq_dbListAdd <listId> <attrSymAddr1> ?<attrSymAddr2>? ...

seq_dbListRemove <listId> <attrSymAddr1> ?<attrSymAddr2>? ...

seq_dbListDestroy <listId>

seq_dbListExtract <listId> <attrSymAddr1> ?<attrSymAddr2>? ...

seq_dbListPut <listId> <attrSymAddr1> <value1> ?<attrSymAddr2> <value2>? ...

seq_dbListList ?<environment>?


DESCRIPTION
seq_dbListCreate:
  create an empty list of attributes
  <envName> : environment name of the database to access
  <access>  : read or write access (r for read, w for write)

  seq_dbListCreate will create and initialize the structure used for
  subsequent multi-read/write operations. It returns a listId, which has
  to be used in other seq_dbList* commands to refer to this particular list.

seq_dbListAdd:
  add attributes to the multi-read/write list
  <listId>  : list handle as returned by seq_dbListCreate
  <attrSymAddrN> : symbolic address of the database attribute, including
                   eventual indices but excluding the environment name
                   (as the environment to access was defined already by
                   seq_dbListCreate)
  Remark that there will be no errors produced if the attribute is already
  in the list.

seq_dbListRemove:
  remove attributes from the multi-read/write list
  <listId>  : list handle as returned by seq_dbListCreate
  <attrSymAddrN> : symbolic address of the database attribute to remove,
                   in exactly the same format as it was added initially with
                   seq_dbListAdd.

seq_dbListDestroy:
  delete a multi-read/write list
  <listId>  : list handle as returned by seq_dbListCreate

seq_dbListExtract:
  extract the information of some attribute(s) from the multi-read/write
  list as a result of a read/write operation
  <listId>  : list handle as returned by seq_dbListCreate
  <attrSymAddrN> : symbolic address of the database attribute whose info
                   is wanted, in exactly the same format as it was added
                   initially with seq_dbListAdd.
  seq_dbListExtract will return a list with the following info for each
  attribute given as an argument:
    - the attribute-type (either SCALAR, VECTOR or TABLE)
    - the list of data-types; data-types are integers reflecting the type
      of the attribute's fields.  For SCALARs and VECTORs this is a single
      value (hence not surrounded by curly braces)
    - the complection status of the multi-read/write operation for this
      particular attribute; 0 means success, 1 signals failure
    - the quality of the data: either OK, SUSPECT, ERROR or DISABLED; for
      an explanation of the meaning of these words, see the CCS db
      documentation.
    - the list of values.  For SCALARs, this is a single value, which will
      include braces if it contains spaces.  For VECTORs, if the attribute
      covers multiple records, this will be a list of values, one per
      record.  For TABLEs, each list of values for each record will contain
      as values as there are fields specified in the attribute.  This scheme
      is identical to what goes for the seq_dbReadSymbolic command.
  If a single attribute was given as argument, the return-string will be a
  list with the 5 elements as described above.  If multiple attributes are
  passed, the return-string will be a list with one element per attribute,
  and each element is on its turn a list with the 5 elements given above.

seq_dbListPut:
  prepare the listId's internal buffer corresponding to a certain attribute
  for a multi-write operation.
  <listId> : list handle as returned by seq_dbListCreate
  <attrSymAddrN> : symbolic address of the attribute
  <valueN> : list of values for this particular attribute, in the format
             as returned by seq_dbReadSymbolic.  I.e. for attributes of
             type VECTOR and TABLE, it is a list with as many elements as
             records have to be written.  As usual in Tcl/Tk lists, if only
             one record is written, i.e. if this list contains only one
             element, enclosing braces are not needed.

  seq_dbListPut will take the values passed, and store them in the memory
  assigned to this attribute, so it is ready to be written out with a
  seq_dbMultiWrite command.  There is no database access involved with
  this command.

seq_dbListList:
  return a list with one element per listId.  Each such element contains
  the following information, in the order as listed:
    - the list handle
    - the environment name it is linked to
    - the use-flag for this list ("r" for read, "w" for write)
    - a list containing the attributes of this multi-read/write list.
  The optional <environment> argument allows to restrict the list to this
  particular environment.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK return if no errors occurred.  The result string is empty, except for
    seq_dbListCreate, seq_dbListExtract and seq_dbListList, where it will
    contain the strings as indicated above.
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - 'wrong # args : should be "seq_dbListCreate <envName> <access>"': the
    command was given with the wrong number of arguments
  - 'wrong # args : should be "seq_dbListAdd <listId> <attrSymAddr1>
```

?<attrSymAddr2>? ..."': the command was given with the wrong number of arguments

- 'wrong # args : should be "seq_dbListRemove <listId> <attrSymAddr1> ?<attrSymAddr2>? ..."': the command was given with the wrong number of arguments
- 'wrong # args : should be "seq_dbListDestroy <listId>"': the command was given with the wrong number of arguments
- 'wrong # args : should be "seq_dbListExtract <listId> <attrSymAddr1> ?<attrSymAddr2>? ..."': the command was given with the wrong number of arguments
- 'wrong # args : should be "seq_dbListPut <listId> <attrSymAddr1> <value1> ?<attrSymAddr2> <value2>? ..."': the command was given with less than 3 arguments or an even number of arguments
- 'wrong # args : should be "seq_dbListList ?<environment>?"': the command was given with the wrong number of arguments
- 'wrong value for access: "<access>" should be either "r" or "w"'
- 'cannot create a multi-read/write list for env <envName>, access <access>': the call to dbListCreate returned an error
- 'invalid list handle: "<listId>"': <listId> does not start with the word "list"
- 'invalid list handle: "<listId>" (no multi-read/write items associated to it)': the shell does not know of any multi-read/write list with this particular <listId>
- 'cannot get info on "<attrSymAddrN>"': during seq_dbListAdd, the call to dbGetAttrInfo() failed, most likely because this symbolic address is not correct.
- 'cannot add <attrSymAddrN> to multi-read/write list "<listId>": the call to dbListAdd() failed for this symbolic address
- 'cannot remove <attrSymAddrN> from multi-read/write list "<listId>"': this symbolic address is not a member of the multi-read/write list, or the call to dbListRemove() failed for other reasons.
- 'cannot extract info of <attrSymAddrN> from multi-read/write list "<listId>"': this symbolic address is not a member of the multi-read/write list, or the call to dbListExtract() failed for other reasons.
- 'cannot destroy multi-read/write list "<listId>"': the call to dbListDestroy() returned an error
- errors from list-splitting (<valueN> for VECTOR and TABLE attributes)
- 'amount of data incompatible with attribute info' during execution of seq_dbListPut, for VECTORs and TABLEs with less/more data than the number of records indicates
- 'record <i>: expected <m> data values; have got <n>': for TABLES the number of values in each sublist (record) of <value> should equal the number of dataType elements.
- 'record <i> field <m>: could not convert "<valueN>" to dbTYPE <n>': the routine dbStrToDe(3) choked on this conversion of a TABLE data element.
- 'record <i>: could not convert "<valueN>" to dbTYPE <n>': as above, but here for the conversion of a VECTOR record.
- 'could not convert "<valueN>" to dbTYPE <n>': as above, but now for the conversion of a SCALAR.

CAUTIONS

Comparing seq_dbLisExtract's return-string for the case of a single attribute with the case of multiple attributes, one might detect a slight inconsistency. For the latter, the reply will consist of a list with one element per attribute. Each of these elements are on their turn a list containing the attribute-type, data-type(s), status, quality and value(s) - some of which can again be sublists.

If there is on the other hand only one attribute passed to seq_dbListExtract, the reply will not be a list with a single element (extrapolating the above scheme), but rather the contents of that single element. The advantage of this approach is that one does not need so many layers of list-handling commands in the most trivial case of a single attribute.

EXAMPLES
(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_dbListCreate wte67 r
list0
seqWish>set attribute1 :PARAMS:SCALARS.scalar_float
seqWish>set attribute2 :PARAMS:VECTORS.vector_double(1:2)
seqWish>set attribute3 :PARAMS:TABLES.full_table(1:2,3:4)
seqWish>seq_dbListAdd list0 $attribute1 $attribute2
seqWish>....
seqWish>seq_dbListRemove list0 $attribute2
seqWish>seq_dbListList
{list0 wte67 r :PARAMS:SCALARS.scalar_float}
seqWish>seq_dbListAdd list0 $attribute3
seqWish>....
seqWish>seq_dbMultiRead list0
seqWish>seq_dbListExtract list0 $attribute1 $attribute3
{{SCALAR 8 0 OK 5.75} {TABLE {8 16} 0 OK {{4 5} {3 4}}}}
seqWish>....
seqWish>seq_dbListPut list0 $attribute1 321.5 $attribute2 {3 7}
seqWish>seq_dbListPut list0 $attribute3 {{1.2 {hi there}} {2. hi!}}
seqWish>seq_dbMultiWrite list0
seqWish>....
seqWish>seq_dbListDestroy list0
seqWish>.....
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbListCreate(3), seqDbListAdd(3), seqDbListRemove(3),
seqDbListDestroy(3),

- - - - - -
Last change: 28/03/02-10:36

seq_dbListDestroy(n)

See seq_dbListCreate(n).

- - - - - -

Last change: 28/03/02-10:36

seq_dbListList(n)

See seq_dbListCreate(n).

- - - - - -

Last change: 28/03/02-10:36

seq_dbListPut(n)

See seq_dbListCreate(n).

- - - - - -

Last change: 28/03/02-10:36

seq_dbListRemove(n)

See seq_dbListCreate(n).

- - - - - -

Last change: 28/03/02-10:36

seq_dbLockPoint(n)

```
NAME
seq_dbLockPoint, seq_dbUnlockPoint - lock/unlock a database point


SYNOPSIS
seq_dbLockPoint <pointName> ?<timeout>?

seq_dbUnlockPoint <pointName>


DESCRIPTION
seq_dbLockPoint:
    will lock a single point in the database.

    <pointName> : symbolic name or alias of the point to lock
    <timeout>   : number of seconds the point will be locked (range: 3 to
                  60); default: 3

    The point will remain locked for <timeout> seconds, unless a
    seq_dbUnlockPoint command is given before the timeout expires

seq_dbUnlockPoint:
    will unlock a locked database point.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURNVALUES
OK return if no errors occurred.  The result string is empty.
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - "wrong # args : should be "seq_dbLockPoint <pointName> ?<timeout>?":
    the command was given with the wrong number of arguments;
  - "wrong # args : should be "seq_dbUnlockPoint <pointName>": the command
    was given with the wrong number of arguments;
  - error message left by Tcl_GetInt applied on <timeout>;
  - "cannot lock point "<pointName>": dbLockPoint returned an error for
    this argument
  - "cannot unlock point "<pointName>": dbUnlockPoint returned an error


CAUTIONS
If the point is already locked, this command will hang until the point
has been released.


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>seq_dbLockPoint "<alias>eric"
seqWish>seq_dbread "<alias>eric.my scalar"
199.5
seqWish>seq_dbUnlockPoint "<alias>eric"
seqWish>.....
```

```
SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbLockPoint(3), seqDbUnlockPoint(3)




- - - - - -
Last change:  28/03/02-10:36
```

seq_dbMultiRead(n)

```
NAME
seq_dbMultiRead, seq_dbMultiWrite -
     commands for the handling of db multi-read/write operations


SYNOPSIS
seq_dbMultiRead <listId1> ?<listId2>? ...

seq_dbMultiWrite ?-block? <listId1> ?<listId2>? ...


DESCRIPTION
seq_dbMultiRead:
  read a list of attributes on a particular environment
  <listIdN> : list handle as returned by seq_dbListCreate

  seq_dbMultiRead will do the actual database access for all attributes
  in the multi-read/write list <listIdN>, which was created and prepared
  previously with the seq_dbList* commands.  The actual values read can
  be extracted with the seq_dbListExtract command.

seq_dbMultiWrite:
  write a list of attributes on a particular environment
  -block : option that signals that all attributes should be considered
           a unit, and consequently to succeed all addresses must be
           correct and writable by the user. Default: no blocking
  <listIdN> : list handle as returned by seq_dbListCreate

  seq_dbMultiWrite will do the actual database access for all attributes
  in the multi-read/write list <listIdN>, which was created and prepared
  previously with the seq_dbList* commands.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK return if no errors occurred.  The result string is empty.
ERROR return if one of the following conditions is met (with corresponding
error message here between single quotes):
  - 'wrong # args : should be "seq_dbMultiRead <listId1> ?<listId2>? ..."':
    the command was given with the wrong number of arguments
  - 'wrong # args : should be "seq_dbMultiWrite ?-block? <listId1>
    ?<listId2>? ..."': the command was given with the wrong number of
    arguments
  - 'invalid list handle: "<listId>"': <listId> does not start with the
    word "list"
  - 'multi-read/write list "<listId>" is currently undefined': the shell
    does not know of any multi-read/write list with this particular
    <listId>
  - 'multi-read for list "<listId> failed': the call to dbMultiRead failed.
  - 'multi-write for list "<listId> failed': the call to dbMultiWrite
    failed


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)
```

```
.....
seqWish>seq_dbListCreate wte67 r
list0
seqWish>set attribute1 :PARAMS:SCALARS.scalar_float
seqWish>set attribute2 :PARAMS:VECTORS.vector_double(1:2)
seqWish>set attribute3 :PARAMS:TABLES.full_table(1:2,3:4)
seqWish>seq_dbListAdd list0 $attribute1 $attribute2 $attribute3
seqWish>....
seqWish>seq_dbListPut list0 $attribute1 321.5 $attribute2 {3 7}
seqWish>seq_dbListPut list0 $attribute3 {{1.2 {hi there}} {2. hi!}}
seqWish>seq_dbMultiWrite list0
seqWish>....
....
```

```
SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbListCreate(3), seqDbListAdd(3), seqDbListRemove(3),
seqDbListDestroy(3),
```

```
- - - - - -
Last change:  28/03/02-10:36
```

```
seq_dbMultiWrite(n)
```

See seq_dbMultiRead(n).

Last change: 28/03/02-10:36

seq_dbReadSymbolic(n)

```
NAME
seq_dbReadSymbolic - read attribute values from on-line database points


SYNOPSIS
seq_dbReadSymbolic ?-noBadQLog? <attrSymAddress> ?<badQualityFlag>?


DESCRIPTION
seq_dbReadSymbolic will read a value of an attribute

-noBadQLog : an optional flag indicating that Quality Errors should not be
             sent to the logging system (which they are by default).  Remark
             that polling of "bad" attributes tends to fill up the log.  If
             -noBadQLog is set, the corresponding error stack will be reset
             (i.e. it will be lost).
<attrSymAddress> : string containing a symbolic address of the attribute
<badQualityFlag> : name of a Tcl boolean variable that will be set/reset
                   according to whether the quality is suspect (bad) resp.
                   OK.  This is useful for scanned attributes only.

seq_dbReadSymbolic will convert the value read to a string, and return it
as the result-string.
Scalar-values are not seen as list elements, i.e. they are never returned
with enclosing braces, unless they are of type string and the braces are
part of that string.
Vector-records are returned as a list and tables as a list of lists (with
each sub-list containing the values of all fields of 1 record). Surrounding
curly braces are added to these sub-lists for table-reads with more than
one field.  They will also appear around every list-element which is a
string containing spaces. I.e. the use of lists is limited to the cases
where they are really needed.  See also the 'CAUTIONS' section.

For floats etc. the precision is determined by tcl_precision.


ENVIRONMENT
tcl_precision:
  global Tcl variable whose value is passed on to dbDeToStr(3) as the
  argument determining the precision (i.e. the maximum number of
  significant digits) of the conversion to string.


RETURN VALUES
OK if no errors occurred.  In this case the result-string contains the
    db value(s)
ERROR if one of the following conditions is met (with the corresponding
error message here between quotes):
  - 'wrong # args: should be "seq_dbReadSymbolic ?-noBadQLog?
    <attrSymAddress> ?<badQualityFlag>?"': the command was given with the
    wrong number of arguments
  - 'wrong option: if "seq_dbReadSymbolic" has 3 args, one of them must be
    "-noBadQLog"': as the message says
  - 'cannot read "<attrSymAddress>", even after resizing buffer' if
    dbReadSymbolic still returned an error even after the receiving buffer
    was resized properly (RTAP only)
  - 'cannot read "<attrSymAddress>"' if dbReadSymbolic returned an error
    from which seq_dbReadSymbolic cannot recover


CAUTIONS
```

The <badQualityFlag> can of course not be set to "-noBadQLog" - but variable names should never start with a dash!

Comparing the return-string for the case of a single record, single field (like SCALAR) with the case of multiple records, one might detect a slight inconsistency. For the latter, the reply will consist of a list with one element per record. Each of these elements will on their turn be a list if multiple fields are involved.
If there is on the other hand only one record to read, the reply will not be a list with a single element (extrapolating the above scheme), but rather the contents of that single element, i.e. the value(s) of the field(s).
So a table with one record, one field gives a result-string containing a single value. One record, multiple fields gives a list with one element per field (each element containing a field's value). Multiple records, one field gives a similar list. Multiple records, multiple fields result in a list with one element per record. Each of these elements are on their turn a list, with one element per field.
The advantage of this approach is that one does not need so many layers of list-handling commands in the most trivial case of a single value.

The internal buffer needed to read the attribute's value is automatically increased if necessary. This memory is not returned to the system until the shell is stopped.

EXAMPLES
(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_dbReadSymbolic ":PARAMS:SCALARS.scalar_float" bad
12.345679
seqWish>set bad
0
seqWish>set tcl_precision 3
seqWish>seq_dbReadSymbolic ":PARAMS:SCALARS.scalar_float"
12.3
seqWish>set tcl_precision 10
seqWish>seq_dbReadSymbolic ":PARAMS:VECTORS.vector_double(2:4)"
12.34567890 0. 0.987654321
seqWish>seq_dbReadSymbolic ":PARAMS:TABLES.full_table(0:1)"
{{First field} second third} {wow {again wow} {more wow}}
seqWish>....
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbReadSymbolic(3), seqInit(3), dbReadSymbolic(3), dbDeToStr(3)

- - - - - -
Last change: 28/03/02-10:36

seq_dbSetCwp(n)

```
NAME
seq_dbSetCwp, seq_dbGetCwp - set/get the current working point


SYNOPSIS
seq_dbSetCwp <pointName>

seq_dbGetCwp ?<envName>?


DESCRIPTION
seq_dbSetCwp:
    will set the current working point

    <pointName> : the database point to set the current working directory
                  to; is either a symbolic address or its alias.

seq_dbGetCwp:
    will return the current working point

    <envName> : name of the environment whose current working point has to
                be returned; defaults to the current environment.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK return if no errors occurred.  For seq_dbGetCwp the result string
    contains the current working point (not including the environment
    name).
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - "wrong # args : should be "seq_dbSetCwp <pointName>": the corresponding
    Sequencer command was given with the wrong number of arguments;
  - "wrong # args : should be "seq_dbGetCwp ?<envName>?": the
    corresponding  Sequencer command was given with the wrong number of
    arguments;
  - "cannot set CWP to "<pointName>": dbSetCwp returned an error
    for this argument
  - "cannot get CWP from "<envName>": dbGetCwp returned an error


CAUTIONS
none


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>seq_dbSetCwp "<alias>eric"
seqWish>seq_dbGetCwp
:users config:eallaert
seqWish>.....


SEE ALSO
```

```
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbSetCwp(3)




- - - - - -
Last change:  28/03/02-10:36
```

seq_dbUnlockPoint(n)

```
See  seq_dbLockPoint(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_dbWriteSymbolic(n)

```
NAME
seq_dbWriteSymbolic - write a value to an attribute given its symbolic
                      address


SYNOPSIS
seq_dbWriteSymbolic <attrSymAddress> ?<attrInfo>? <value>


DESCRIPTION
seq_dbWriteSymbolic will write the <value> into an attribute pointed to by
its symbolic address.  The arguments are:

<attrSymAddress> : string containing the symbolic address of the attribute

<attrInfo> : a list containing 4 elements, in the following order:
  - the type of the attribute, i.e. SCALAR, VECTOR or TABLE (attrType)
  - the data type(s) of the attribute; integer(s) in the dbTYPE range
    (datatype).  Remark that for TABLEs, only the data types of the fields
    which will be written into must be specified.
  - the number of attribute records to write
  - the number of attribute fields to write
  The latter two pieces of informations correspond to the information
  returned by seq_dbGetAttrInfo.
  If <attrInfo> is not specified, a call to dbGetAttrInfo will be issued
  to retrieve these data (which may slow down this function somewhat).

<value>: the new value(s) for the attribute.

For attributes of type VECTOR and TABLE, the last argument is a list with
as many elements as records have to be written.  As usual in Tcl/Tk lists,
if only one record is written, i.e. if this list contains only one element,
enclosing braces are not needed.

For attributes of type TABLE, the data types (part of <attrInfo>) is a
sublist.  This data type list contains one element per field.  The
argument with the values consists of one element per record, as for
VECTORs. However, here each element is a sublist, with one element (value)
per field.

The number of records to write for VECTORs and TABLEs is taken from the
<attrInfo>, or, in its absence, via a call to dbGetAttrInfo.  It is checked
against the length of the list of new attribute values.

The DB end-range character '$' is fully supported for both record and field
ranges, as are record addressing by content and field addressing by name.

Remark that some values (e.g. time values) can be (sub)lists.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK if no errors occurred, with an empty result string.
ERROR if one of the following conditions is met (with corresponding
error message here between quotes):
  - 'wrong # args: should be "seq_dbWriteSymbolic <attrSymAddress>
    ?<attrInfo>? <value>"': the corresponding Sequencer command was given
    with the wrong number of arguments;
```

- '<attrInfo> should contain the data type' if the <attrInfo> list has only one element;
- '<attrInfo> for VECTOR should contain the record count' if the <attrInfo> list has only two elements for an attribute of type VECTOR;
- '<attrInfo> for TABLE should contain the record and field counts' if the <attrInfo> list has less than 4 elements for an attribute of type TABLE;
- 'attribute type "<attrType>" unknown; should be SCALAR, VECTOR or TABLE': the attribute type specified is none of the three listed
- errors from parsing integers (data type, record count, field count)
- errors from list-splitting (<attrInfo>; dataType for TABLE; <value> for VECTOR and TABLE)
- 'amount of data incompatible with attribute info' for VECTORs and TABLEs with less/more data than the number of records indicates
- 'amount of supplied data type info inconsistent with number of fields' for TABLEs with less/more data type info than the number of fields indicates
- 'record <i>: expected <m> data values; have got <n>': for TABLES the number of values in each sublist (record) of the last argument should equal the number of dataType elements.
- 'record <i> field <m>: could not convert "<valueX>" to dbTYPE <n>': the routine dbStrToDe(3) choked on this conversion of a TABLE data element.
- 'record <i>: could not convert "<valueX>" to dbTYPE <n>': as above, but here for the conversion of a VECTOR record.
- 'could not convert "<valueX>" to dbTYPE <n>': as above, but now for the conversion of a SCALAR.
- 'cannot write to database': the call to dbWriteSymbolic failed

CAUTIONS

Although "named" record and field indices are allowed, they require an additional access to the database to convert them to numbers. This means extra overhead, which is avoided by giving numbers. Additionally, this works only for the local environment, as it relies on dbGetDirAddr(3).

The buffer needed to write the attributes' values to the database via an dbWriteSymbolic call is automatically resized if necessary, and never deallocated. This may cause for unfairly big writes to TABLEs a one-time noticeable growth of the process memory.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_dbWriteSymbolic <alias>me.myScalar {SCALAR 5 1 1} 327
seqWish>seq_dbWriteSymbolic <alias>me.myVector(1:2) {VEC 6 15 1} {3 7}
attribute type "VEC" unknown; should be SCALAR, VECTOR or TABLE
seqWish>seq_dbWriteSymbolic <alias>me.myVector(1:2) {VECTOR 6 15 1} {3 7}
seqWish>set a [seq_dbReadSymbolic <alias>me.myTable(0:0)]
seqWish>seq_dbWriteSymbolic <alias>me.myTable(1:1) {TABLE {20 25} 5 2} $a
seqWish>seq_dbWriteSymbolic <alias>me.myTable(1:1,1:1) {TABLE 25 5 2} ""
seqWish>....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDbWriteSymbolic(3), dbStrToType(3), dbWriteSymbolic(3) , dbGetDirAddr(3)

```
- - - - - -
Last change:  28/03/02-10:36
```

seq_deleteHandle(n)

NAME

seq_deleteHandle - delete a handle and release related memory

SYNOPSIS

seq_deleteHandle <handle>

DESCRIPTION

seq_deleteHandle will delete a <handle> and its related resources. It provides a unified way to do this for any of the handles currently used by the Sequencer shell:

- a command handle, as obtained by issuing a "seq_msgSendCommand" (in this section, the word "command" refers to such a message sent with "seq_msgSendCommand). The handle itself will be removed; all buffers corresponding to replies which are already in, but which are not yet retrieved via e.g. a seq_msgRecvReply command, will be freed. Remark that also the corresponding commandId will be freed up for re-use. Consequently, replies for this particular command which are not yet on the queue can no longer be handled properly after issuing a seq_deleteHandle (see CAUTIONS).

- an event handle, as obtained by issuing a "seq_evtAttach" command. There will be an evtDetach() call before the handle is removed. This is equivalent to the command "seq_evtDetach".

- an alarm handle, as obtained by issuing a "seq_alrmAttach" command. There will be an alrmDetach() call before the handle is removed. This is equivalent to the command "seq_alrmDetach".

- a script handle, as received with a SCRIPT command (see seq_msgSendReply). The process which sent this SCRIPT command will receive a final reply indicating that the script handle is being deleted, and no further processing will take place.

- a multi-read/write list, as obtained with a seq_dbListCreate command. This is identical to issuing the seq_dlListDestroy command.

As indicated above, for event-handles and list-handles "seq_deleteHandle" is equivalent to "seq_evtDetach" resp. "seq_dbListDestroy". There is no obvious reason why one would want to delete a script-handle (as they are under normal conditions short-lived and anyway not numerous), but for reasons of uniformity this type of handles can be dealt with as well. So the objects of seq_deleteHandle might remain limited to the command-handles.

ENVIRONMENT

no environment nor Tcl variables are read nor set

RETURN VALUES

OK return if no errors occurred. The result string is empty.

ERROR return if one of the following conditions is met (with corresponding error message here between single quotes):

- 'wrong # args : should be "seq_deleteHandle <handle>"' the corresponding Sequencer command was given with the wrong number of arguments;
- 'invalid handle: "<handle>"' if the handle is not one of the four types listed above (looking at the first few characters)

```
  - 'invalid cmd handle: "<handle>" (no replies pending on it)': if the
    command-handle does not have any replies pending on it
  - 'invalid event handle: "<handle>" (no database-events associated to it)'
    if the  event-handle does not have any attributes attached
  - 'invalid script handle: "<handle>" (no SCRIPT command related to it)'
  - 'invalid list handle: "<handle>" (no multi-read/write items associated
    to it)'
```

CAUTIONS

The deletion of a command-handle should be exercised with proper care. The reason being that the process who received the associated command will not be informed that it should no longer send replies to this command. If it still does, it will use a commandId which the Sequencer shell in the meantime may have allocated to another command. This could obviously lead to confusion in the reply handling. So this deletion of a command-handle is intended for cases where one is confident that the partner process is no longer alive, or when one has other means to prevent the partner from sending "pending" replies.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>set cmdId [seq_msgSendCommand $env $proc INIT]
.....    (user notifies that $proc died during initialization)
seqWish>seq_deleteHandle $cmdId
seqWish>.....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqDeleteHandle(3), deleteHandle(3)

- - - - - -
Last change: 28/03/02-10:36

seq_errAdd(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errCloseStack(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errDisplay(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errGetFromStack(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errGetStackSize(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errLog(n)

NAME

seq_errLog - log a single Sequencer error

SYNOPSIS

seq_errLog <error> <srcId>

DESCRIPTION

seq_errLog will handle the logging of a single error for the Sequencer, i.e. it will first reset the error stack, then log <error> and finally close (and thereby reset) the error stack.

<error> : string containing a description of the error
<srcId> : string containing the source identifier

ENVIRONMENT

no environment nor Tcl variables are read nor set

RETURN VALUES

OK return if no errors occurred. The result string is empty.
ERROR return if seq_errAdd fails, with a corresponding error message as generated by seq_errAdd.

CAUTIONS

none

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_errLog "Who the hell deleted this file??" "file_proc"
seqWish>.....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seq_errResetStack(n), seq_errAdd(n), seq_errCloseStack(n)

- - - - - -
Last change: 28/03/02-10:36

seq_errPrint(n)

```
See  seq_errResetStack(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_errResetStack(n)

```
NAME
seq_errResetStack, seq_errAdd, seq_errCloseStack, seq_errPrint,
seq_errDisplay, seq_errGetStackSize, seq_errGetFromStack -
         interface to the CCS error logging


SYNOPSIS
seq_errResetStack ?-internal|-reply?

seq_errAdd ?-internal|-reply? <error> <sourceId>

seq_errCloseStack ?-internal|-reply?

seq_errPrint ?-internal|-reply?

seq_errDisplay ?-internal|-reply? ?-suspend?

seq_errGetStackSize ?-internal|-reply?

seq_errGetFromStack ?-internal|-reply? <frameNr>


DESCRIPTION
These commands will handle the logging of errors for the Sequencer (see
further below for a description of the -internal and -reply options):

seq_errResetStack: resets the error stack; to be used after the
  successful recovery from an error, or at initialization time, before
  calling seq_errAdd for the first time (see also CAUTIONS section).

seq_errAdd adds the string <error> to the error stack, pointing to
  <sourceId> (a string) as the originator of the error.  The moduleId
  (default: "seq") with which it will be logged can be modified via the
  global Tcl variable seq_moduleId.  Of course this would then require that
  a proper error definition file exist for this new moduleId.  In
  particular, error number 1 is reserved for this command; it needs to have
  been defined with the format-string ".256s".

seq_errCloseStack logs the last added error and closes the stack; used
  if no error recovery is possible.  Remark that after this the error
  stack is reset automatically.

seq_errPrint will return a list with all the elements of the last CCS
  error, which was  either produced internally or received as a reply.
  The order of the list is as follows (see also a description of the CCS
  error structure for more details):
      o environment name
      o stackId structure (hostId and localNumber) as a list
      o stack sequence number
      o location identifier
      o module identifier
      o error number (must be > 0)
      o severity (W for Warning, S for Serious, F for Fatal)
      o error message (between braces if it contains spaces)
      o timestamp of the error

seq_errDisplay will pop up a panel showing the stack of the last CCS
  error, which was either produced internally or received as a reply.  If
  the -suspend option is specified, the Sequencer script will wait for
  this error panel to be closed before continuing.
```

seq_errGetStackSize returns the size of the stack of the last CCS error; 0 means it is empty

seq_errGetFromStack will return the contents of the frame <frameNr> in the error stack as a list, formatted similar as seq_errPrint

The Sequencer uses 3 different error stacks, which can be selected via the presence or absence of the options -internal and -reply. Also the logging of these stacks can be influenced, via the setting of the Tcl-variable seq_errLogging. The use of this variable, and the selection plus purpose of the different stacks is explained in the next few paragraphs.

Whenever the Sequencer gets a CCS-error it cannot recover from, the internal error stack will be submitted to an errCloseStack call (seq_errLogging set to auto). This means that the Sequencer script does not need to include any instructions to close and log any error stack. In this case, the call to errCloseStack() will actually be preceded by an errAdd() call, identifying the Sequencer as the top-level.

If the Sequencer receives via seq_msgRecvReply an error reply, this reply error stack will also be logged automatically (if the default setting of the variables seq_errLogging and seq_errReplyMerging is used, see below), however without the Sequencer appending its own message. If you want to add your own error messages to this reply stack, you can set seq_errLogging to "manual" and then give the proper seq_errAdd / seq_errCloseStack commands using the option -reply. Alternatively, one can use the seq_errReplyMerging variable to have this stack automatically merged into one of the two other stacks (see below).

In both cases, a copy of this error stack is kept in the "display stack", so that it still can be displayed with e.g. the seq_errGet* commands even after errCloseStack() was called automatically (which resets the stack, rendering its previous contents unaccessible).

For seq_errResetStack, seq_errAdd and seq_errCloseStack the error stack referred to is completely independent from the Sequencer's internal error stack (used for returned CCS errors, as described above), unless the -internal option is specified. This option is intended for the cases where one does not want automatic logging, by setting the global Tcl-variable seq_errLogging to manual - see the description of seq_errLogging below. Remark however that issuing a seq_errCloseStack command with seq_errLogging set to manual behaves the same as for auto, i.e. the error stack will be logged, and it will be copied into the display stack" for further display purposes.

The error stack referred to by seq_errPrint, seq_errDisplay, seq_errGetStackSize and seq_errGetFromStack is by default the "display stack", i.e. a copy of either the last error message received by seq_msgRecvReply, the last internal error logged automatically or the last error stack logged explictly via seq_errCloseStack. Again, the -internal and -reply options are only useful when the global Tcl-variable seq_errLogging is set to manual. The -reply option is to refer to the last error which was returned as a reply to a command. This is useful in cases where due to further script execution the "display stack" may get overwritten by internal errors.

If both the -internal and -reply options are given, the one given last prevails.

ENVIRONMENT

The environment variable NO_ERR_DISPLAY affects the behaviour of seq_errDisplay, in that it will send its output to stdout instead of popping up a new window if set. See also errDisplay(3).

The global Tcl variable seq_errLogging reflects how the logging of errors occuring during the execution of seq_* commands are dealt with; "auto" (default) leads to automatic logging of all unrecoverable errors (by issuing an errCloseStack()), "off" will not log any error (by issuing an errResetStack), while "manual" will not issue any err*Stack command.In the latter case it is assumed that the script will take care of the error stack via "seq_err* -internal|-reply" commands. In all cases, the corresponding error stack will be copied over to the "display stack".

The global Tcl variable seq_errReplyMerging determines how the merging of error-replies will take place: "internal" will append the reply-stack to the internal error stack, "local" will make the reply-stack append to the default stack of seq_errAdd etc, while "off" (default) leaves the error reply stack alone. In the former 2 cases, the error reply stack will be reset after merging, without any logging. In the latter case, it will get logged automatically if seq_errLogging is set to "auto", in which case it will also be copied over to the "display stack".

The global Tcl variable seq_moduleId has its effect on how errors are logged with seq_errAdd - see above.

RETURN VALUES

OK return if no errors occurred. The result string is empty.
ERROR return if one of the following conditions is met (with corresponding error message strings here printed in boldface):

```
  - wrong # args : should be "seq_errResetStack ?-internal|-reply?"
    wrong # args : should be "seq_errAdd <error> ?-internal|-reply?
    <sourceId>"
    wrong # args : should be "seq_errCloseStack ?-internal|-reply?"
    wrong # args : should be "seq_errGetStackSize ?-internal|-reply?"
    wrong # args : should be "seq_errGetFromStack ?-internal|-reply?
                    <frameNr>"
    wrong # args : should be "seq_errPrint ?-internal|-reply?"
    wrong # args : should be "seq_errDisplay ?-internal|-reply?
                    ?-suspend?"
    the corresponding Sequencer command was given with the wrong number
    of arguments;
```

- sourceId "<sourceId>" too long if the string length of <sourceId> exceeds the limits imposed by the ccsERROR structure;

- error message left by Tcl_GetInt, when trying to interpret <frameNr> as an integer;

- could not do an errAdd - check the log if during the processing of the seq_errAdd command the call to errAdd(3) failed.

- could not get error-frame <frameNr> from stack if seq_errGetFromStack returns an error (e.g. <frameNr> exceeds the current stack size)

- wrong option: <option> should be "-internal", "-reply" or "-suspend" if any of these words were misspelled for the seq_errDisplay command

CAUTIONS

Under NOCCS, only seq_errResetStack, seq_errAdd and seq_errCloseStack are avaliable.

Under CCS-lite, the current setting of RTAPENV is taken whenever the first

CCS-function gets called. If this first CCS-function is not a ccsInit(), this value of RTAPENV will anyway be used later when ccsInit() get called, independent of RTAPENV's value at that later time. This means that e.g. in the wtctest environment, issuing first a seq_errResetStack command followed by a set env(RTAPENV) xyz and a seq_ccsInit command, will lead to a ccsInit() in the wtctest environment instead of the xyz environment. This is not inherent to seq, but rather to CCS-lite. In fact, to enable an effective modification of the environment variable RTAPENV within a Sequencer script (e.g. based on runstring options), the Sequencer's internal initialization does not call any CCS-function under CCS-lite, thereby shifting the responsibility to do things in the proper order to the application.

seq_errGetStackSize and seq_errGetFromStack manipulate either the reply or the internal error stack. The stack set aside for seq_errAdd, seq_errCloseStack and seq_errResetStack is a different one, and hence cannot be seen by the seq_errGet* commands.

The arguments of seq_errAdd will all be logged as errors with a severity of "warning".

seq_errDisplay will by default use the information of the last CCS error, which is stored in a global C-variable for use by all seq-commands. So if the seq_errDisplay command is not called immediately after detecting such an error, or if the -suspend option is not used, the displayed values may not correspond to what is expected.
Subsequent calls to seq_errDisplay, without destroying windows in between, will simply update the information displayed in these error stack windows. This may not be the desired effect for panels with classes that can try simultaneously to display errors via the seq_errDisplay command.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_errResetStack
seqWish>seq_errAdd "Who the hell deleted this file??" "my procedure"
seqWish>seq_errCloseStack
...(This error can now be seen with the log-monitor)...
.....
seqWish>seqSendCommand wte67 otherSeq SCRIPT {puts hello}
seqWIsh>seq_msgRecvReply cmd0 en lr -all
seqERR_NO_CMDS_ALLOWED : Command "SCRIPT" rejected - no commands allowed
seqWish>seq_errGetStackSize
1
seqWish>seq_errGetFromStack 1
wte67 {0 4869} 1 seqMsgDispatch.c seq 122 W {seqERR_NO_CMDS_ALLOWED :
Command "SCRIPT" rejected - no commands allowed} {97-02-14
15:35:53.324656}
seqWish>.....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
errResetStack(3), errAdd(3), errCloseStack(3), errGetStackSize(3), errGetFromStack(3), errDisplay

```
- - - - - -
Last change:  28/03/02-10:36
```

seq_evtAttach(n)

```
NAME
seq_evtAttach, seq_evtDetach, seq_evtSingleDisable, seq_evtSingleEnable,
seq_evtList - database and event management commands


SYNOPSIS
seq_evtAttach <attrName> <filter> <script>
or
seq_evtAttach <cmdId> <timeout> <script>

seq_evtDetach <eventHandle>

seq_evtSingleDisable <eventHandle>

seq_evtSingleEnable <eventHandle>

seq_evtList ?-status? ?-attrName? ?-filter? ?-script? | ?-all?


DESCRIPTION
seq_evtAttach:
  in its first form will attach an event to a database attribute:

    <attrName> : the database attribute to which an event has to be
                 attached; is either a symbolic address or its alias.
    <filter>   : to condition which should generate the event.  It can be:
                 < or LT for less than
                 <=, =< or LE for less or equal
                 =, == or EQ for equal
                 > or GT for greater than
                 =>, >= or GE for greater or equal
                 != or NE for not equal
                 w or W for any write
                 <> or >< for deadband
    <script>   : the script to execute when an event occurs; this script
       will be evaluated at global level.  The following substitutions will
       be performed on this script before evaluation:
       - %A : name of the attribute causing the event
       - %D : event data; this is a list, whose contents depends on the
              attribute type as follows (see also evtParseMsg(3))
              for SCALAR: {<dataType> <oldValue> <newValue>}
              for VECTOR: {<startElement> <endElement>}
              for TABLE: {<startElement> <endElement> <startField>
                            <endField>}
       - %P : info about the process causing the event; a list with two
              elements: {<envName> <processName>}.  Remark however that
              events associated with LCU database attributes do not return
              this information, and an attempt to use this substitution
              will result in errors in this case.
       - %Q : quality (for scalars only).  It gives a list with 2 values:
                  {<oldQuality> <newQuality>}
              where <oldQuality> and <newQuality> can be either OK,
              SUSPECT, ERROR, or DISABLED.  The exact meaning of these
              words can be found in the evtParseMsg(3) manpage.
              For vectors and tables, this will give an empty list.
       - %T : attribute type.  Either SCALAR, VECTOR or TABLE
       - %U : the user-message part of the event
       - %% : a single % character
       Whenever the substitution on the original script or the evaluation
       of the resulting script yields an error, the corresponding event
       will be disabled, and the error will be logged and also signalled in
```

```
        a separate window popping up via Tcl_BackgroundError.

    in its second form will attach the execution of a script to an incoming
    reply:
      <cmdId>: as returned by seq_msgSendCommand

      <timeout>: timeout value in ms; a timeout-error reply will be generated
         if no reply came in within that time.  Zero means no timeout.  The
         absolute value is taken.

      <script>: a script to be executed (at global level) when such a reply
         comes in; this script will be executed once for each reply,
         independent from the value for lastReply or the error condition,
         including timeout.  In other words, the script must intercept these
         different conditions.

         The following substitutions will be performed on the script before
         it is evaluated:
         - %A : ASCII reply received in the message body; if this reply is
                empty or contains any spaces, it will be surrounded by curly
                braces.
         - %E : error number from the received error structure; 0 means no
                error
         - %F : formatted binary reply received in the message body; if this
                reply is empty or contains any spaces, it will be surrounded
                by curly braces.  Remark that this also works fine on ASCII
                replies, although it is not as efficient as %A. Unformatted
                binary replies will be left empty ({}), while replies without
                an entry in the CDT will be treated as ASCII.
         - %L : last reply flag; "0" for not set, "1" if set
         - %R : reply received in the message body; if this reply is empty
                or contains any spaces, it will be surrounded by curly braces.
                This substitution is equivalent to %F if the command was sent
                with command checking on (see seq_msgSendCommand(n));
                otherwise it behaves as %A.
         - %T : timeout flag; "0" for no timeout, "1" for timeout
         - %% : a single % character

         Whenever the substitution on the original script or the evaluation
         of the resulting script yields an error, the corresponding event
         will be disabled.  On top of that, the error will be logged and also
         signalled in a separate window popping up via Tcl_BackgroundError.

      Replies already on the internal stack will be treated as events.

      After the last reply is dealt with, the link between the <cmdId> and
      the <script> is deleted (as in fact the command handle with all
      associated information is removed).

    The distinction between the two forms of the seq_evtAttach command is
    done based on the second argument: if it is an integer, then it must be
    the attachment to an event-reply.

seq_evtDetach:
      will detach a previously attached event from a database attribute,
      and delete the event handler

      <eventHandle>  : the event handle, as returned by seq_evtAttach.

seq_evtSingleDisable:
      will disable a previously attached event

      <eventHandle>  : the event handle, as returned by seq_evtAttach.
```

```
seq_evtSingleEnable:
    will enable a previously attached and disabled event

    <eventHandle>  : the event handle, as returned by seq_evtAttach.

seq_evtList:
    lists the already defined events with their properties as requested:

    -status   : tell whether for each event whether it is enable or disabled
    -attrName: give the attribute name to which the event is attached
    -filter   : include also the filter
    -script   : list the script that will be evaluated when the event occurs
    -all      : shorthand for the above 4 properties

    The result string is contains per event a sublist with the elements as
    requested.  They will be listed in the order as given above, i.e.
    independent of the order of these options on the command line, and
    preceeded by the event handle.  Remark that attrName, filter and script
    are in the order and format as required for  seq_evtAttach.


ENVIRONMENT
The value of the seq_ccsCmdCheck variable at the time the command was sent
can be used to flag the (expected) presence of a CDT of the process the
reply comes from.  It influences the behaviour of the %R substitution.  See
also seq_msgSendCommand(n) and seqInit(3).

The seq_evt* commands have corresponding entries in the global Tcl array-
variable seq_debug, which controls the printing of debug information (see
Seq_Init(3)). On top of that, if bit 3 of seq_debug(seq_msgDispatch) is set,
there will be additional info printed on incoming event messages.


RETURN VALUES
OK return if no errors occurred.  For seq_evtAttach the result string
    contains the event handle (eventNNN for database events, <cmdId> for
    reply events) to use when referring to this this event with any other
    seq_evt* command.  For seq_evtList the result string is a list of
    sublists (one sublist per event, with elements as requested).
ERROR return if one of the following conditions is met (with corresponding
error message here in bold-face):
  - wrong # args : should be "seq_evtAttach <attrName> <filter> <script>"
    or "seq_evtAttach <cmdId> <timeout> <script>" when the seq_evtAttach
    command was given with the wrong number of arguments;
  - invalid cmd handle "<cmdId>" (no replies pending on it) if a command
    was given with a <cmdId> which has no more pending replies (i.e. the
    handle is no longer valid)
  - invalid event handle "<eventHandle>" (no database-events associated
    with it) if a command was given with an <eventHandle> which is no
    longer valid
  - invalid filter: "<filter>" should be one of "< <= = > => != W" when
    the filter is not valid
  - cannot attach event to "<attrName>" if the call to evtAttach(3)
    returns an error
  - wrong # args: should be "seq_evtDetach <eventHandle>" when the
    seq_evtDetach command is given with the wrong number of arguments
  - invalid cmd handle "<cmdId>" if <cmdId> is not a command handle
  - invalid event handle "<eventHandle>" if <eventHandle> is not an event handle
  - cannot detach <eventHandle> if the call to evtDetach(3) returns an error
  - wrong # args: should be "seq_evtSingleDisable <eventHandle>" when the
    seq_evtSingleDisable command is given with the wrong number of
    arguments
```

- cannot de-activate <eventHandle> when seq_evtSingleDisable(3) returns an error
- wrong # args: should be "seq_evtSingleEnable <eventHandle>" when the seq_evtSingleEnable command is given with the wrong number of arguments
- cannot re-activate <eventHandle> when seq_evtSingleEnable(3) returns an error
- wrong # args: should be "seq_evtList ?-status? ?-attrName? ?-filter? ?-script? | ?-all?" when the seq_evtList command is given with the wrong number of arguments
- bad option: "<arg>" should be -status, -attrName, -filter, -script or -all when the seq_evtList command is given with a bad argument

CAUTIONS

As events associated with LCU database attributes do not return the full information about the process causing the event, an attempt to use the %P substitution will result in errors in this case.

A deadband event can only be attached to an LCU database attribute; moreover, this requires the event to be configured previously with a call to evtConfig(3) (which only exists in LCC).

For reply events, the CDT must be loaded into cmdManager in order to be able to format replies. See cmdManager(1) and cmdSetup(1). The CDT is not accessed if "%A" is used instead of "%R".

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
..... (<alias>my.scalar is a float with value 1.23)
seqWish>seq_evtAttach <alias>my.scalar w {set new [lindex %D 2]; set b 5}
event2
seqWish>set b 32
seqWish>.....
..... (<alias>my.scalar gets modified to 32.1)
seqWish>set b
5
seqWish>set new
32.1
seqWish>set b 42
seqWish>seq_evtSingleDisable event2
seqWish>.....
..... (<alias>my.scalar gets modified to 4.3)
seqWish>set b
42
seqWish>set new
32.1
seqWish>seq_evtDetach event2
seqWish>.....
```

SEE ALSO

Tcl and the Tk toolkit, John K. Ousterhout, ISBN 0-201-6337-X
evtAttach(3), evtDetach(3), evtSingleDisable(3), evtSingleEnable(3), seq_msgDispatch(n), seqInit(3)

- - - - - -

```
Last change:  28/03/02-10:36
```

seq_evtDetach(n)

```
See  seq_evtAttach(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_evtList(n)

```
See  seq_evtAttach(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_evtSingleDisable(n)

```
See  seq_evtAttach(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_evtSingleEnable(n)

See seq_evtAttach(n).

- - - - - -

Last change: 28/03/02-10:36

seq_findFile(n)

See seq_relToAbsPath(n).

- - - - - -

Last change: 28/03/02-10:36

seq_fitsDate(n)

See seq_relToAbsPath(n).

- - - - - -

Last change: 28/03/02-10:36

seq_isoTime(n)

See seq_relToAbsPath(n).

- - - - - -

Last change: 28/03/02-10:36

seq_isoTimeToClock(n)

See seq_relToAbsPath(n).

- - - - - -

Last change: 28/03/02-10:36

seq_logData(n)

NAME
seq_logData - store a single message in the local host's logfile

SYNOPSIS
seq_logData <logString>

DESCRIPTION
seq_logData will log a single line. It will be logged with a logId of 0 (general purpose log identifier).

<logString> : string that has to be logged

ENVIRONMENT
no environment nor Tcl variables are read nor set

RETURN VALUES
OK return if no errors occurred. The result string is empty.
ERROR return if one of the following conditions is met (with corresponding error message here between quotes):

```
  - 'wrong # args : should be "seq_logData logString"': the corresponding
    Sequencer command was given with the wrong number of arguments;
  - 'cannot log "<logString>"': logData returned an error for this argument
```

CAUTIONS
None

EXAMPLES
(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_logData "Hi mom!  I crossed the street without looking!"
seqWish>.....
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqLogData(3), logData(3)

- - - - - -
Last change: 28/03/02-10:36

seq_logFitsAction(n)

```
NAME
seq_logFitsAction, seq_logFitsComment, seq_logFitsEvent,
seq_logFitsParRecord  - generate FITS logs


SYNOPSIS
seq_logFitsAction <dictionary> <category> <subsys> <action>

seq_logFitsComment ?-<who>? <logString>

seq_logFitsEvent ?-<type>? <logString>

seq_logFitsParRecord ?-<type>? <dictionary> <cat> <subsys> <param> <value>


DESCRIPTION
seq_logFitsAction will generate a FITS action record. Its arguments are:
  <dictionary> : name of the dictionary that will be checked for
                   correctness of the other arguments; remark that this
                   does not include the "ESO-VLT-DIC" prefix.
  <category>   : 3-letter keyword category
  <subsys>     : subsystem name(s)
  <action>     : verb defining the action; corresponds to parameter-name
                   in the dictionary.

seq_logFitsComment will generate a FITS comment record. Its arguments are:
  <who>   : "classname" of generator of this log; legal values are:
          "observer", "staff", "remote control" or "night assistant". These
          names can also be abbreviated to "ob", "sa", "rc" respectively
          "na", and they can contain any mixture of lower- and uppercase
          characters. If <who> is not specified, a free-format comment will
          be logged.
  <logString> : string that has to be logged

seq_logFitsEvent will generate a FITS event record. Its arguments are:
  <type>   : "unforeseen" (default) or "recovery". These names can also be
          abbreviated to "ufo" resp "rec", and they can contain any mixture
          of lower- and uppercase characters.
  <logString> : string that has to be logged

seq_logFitsParRecord generates a FITS parameter record. Its arguments are:
  <type>       : either "int" (for 32-bit integers), "real" (for doubles) or
                 "string" (for strings containing up to 256 characters)
  <dictionary> : name of the dictionary that will be checked for
                 correctness of the other arguments; remark that this
                 does not include the "ESO-VLT-DIC" prefix.
  <cat>        : 3-letter keyword category
  <subsys>     : subsystem name(s)
  <parameter>  : parameter name (last substring of the short-fits keyword)
  <value>      : value for the parameter (in 256-byte string).

  This seq_logFitsParRecord command is normally used with a <type> option,
  indicating what sort of conversion (from string values) should take place.
  Without this <type> option, seq_logFitsParRecord should be used with
  extreme caution only, as one needs to take care that <value> is (at least)
  256 bytes long. See also the CAUTIONS section below.
  Conversion from string values is done using sscanf(3), which means that
  e.g. floats are simply truncated if converted to integer. No errors are
  returned if this conversion fails (e.g. attempting to convert "abc" to
  a integer), but the value shown in the log will of course be wrong (it
  will be set to zero).
```

The exact format of the FITS logs generated by these commands is described in the manpages of the underlying respective logFits*(3) procedures.

ENVIRONMENT

The setting of the global Tcl variable seq_moduleId is used as the CCS module-identifier in the the underlying logFits*(3) calls.

RETURN VALUES

OK return if no errors occurred. The result string is empty.
ERROR return if one of the following conditions is met (with corresponding error message here between quotes):

- 'wrong # args : should be "seq_logFitsAction <dictionary> <category> <subsys> <action>"': the corresponding Sequencer command was given with the wrong number of arguments;
- 'wrong # args : should be "seq_logFitsComment ?-<who>? <logString>"': the corresponding Sequencer command was given with the wrong number of arguments;
- 'wrong # args : should be "seq_logFitsEvent ?-<type>? <logString>"': the corresponding Sequencer command was given with the wrong number of arguments;
- 'wrong # args : should be "seq_logFitsParRecord ?-<type> <dictionary> <cat> <subsys> <param> <value>"': the corresponding Sequencer command was given with the wrong number of arguments;
- 'cannot generate FITS action log - dictionary <dictionary>, category <category>, subsystem <subsys>, action <action>': the underlying logFitsAction(3) call returned an error.
- 'wrong generator: "<who>" should be "ob|observer", "rc|remote control", "sa|staff" or "na|night assistant"'
- 'cannot generate FITS comment log "<logString>": the underlying call to logFitsComment(3) returned an error.
- 'wrong type: "<type>" should be "ufo|unforeseen" or "rec|recovery"'
- 'cannot generate FITS event log "<logString>": the underlying call to logFitsEvent(3) returned an error.
- 'wrong type: "<type>" should be "int", "real" or "string"'
- 'cannot generate FITS paramater record log - dictionary <dictionary>, category <cat>, subsystem <subsys>, param <param>, value <value>': the underlying logFitsParRecord(3) call returned an error.

CAUTIONS

The seq_logFitsParRecord command calls the logFitsParRecord(3) function, which does a binary copy of 256 bytes of <value>. This means that in the presence of a valid <type> option, there will internally be a conversion, into a 256-byte buffer. Without this <type> option, the user of this command mandatorily needs to take care of the encoding (i.e. put the right number of bytes in the proper order within <value>), and in particular that <value> does contain (at least) 256 bytes. Not following this rule could result in segmentation errors. See see also Tcl's binary(n) command and the example below.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_logFitsComment "This is a free-field FITS comments log"
seqWish>.....
seqWish># the easy way to log an integer value
seqWish>seq_logFitsParRecord -int OBS OBS "" ID $value
seqWish># or the hard way .....
seqWish>if {[cequal $tcl_platform(byteOrder) bigEndian]} {set i I}
seqWish>seq_logFitsParRecord OBS OBS "" ID [binary format ${i}x252 $value]
seqWish>.....
seqWish>seq_logFitsParRecord -string OBS OBS "" NAME "any string"
seqWish>.....
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
logFitsAction(3), logFitsComment(3), logFitsEvent(3), logFitsParRecord(3)

- - - - - -
Last change: 28/03/02-10:36

seq_logFitsComment(n)
See seq_logFitsAction(n).

- - - - - -
Last change: 28/03/02-10:36

seq_logFitsEvent(n)

See seq_logFitsAction(n).

- - - - - -
Last change: 28/03/02-10:36

seq_logFitsParRecord(n)

See seq_logFitsAction(n).

- - - - - -

Last change: 28/03/02-10:36

seq_logIntParRecord(n)

See seq_logStringParRecord(n).

- - - - - -

Last change: 28/03/02-10:36

seq_logRealParRecord(n)

See seq_logStringParRecord(n).

- - - - - -

Last change: 28/03/02-10:36

seq_logStringParRecord(n)

```
NAME
seq_logStringParRecord, seq_logIntParRecord, seq_logRealParRecord -
FITS parameter logging for strings, integers and floats


SYNOPSIS
seq_logStringParRecord <dictionary> <category> <subsys> <parameter> <value>

seq_logIntParRecord    <dictionary> <category> <subsys> <parameter> <value>

seq_logRealParRecord   <dictionary> <category> <subsys> <parameter> <value>


DESCRIPTION
These commands are all simple wrappers around seq_logFitsParRecord, ensuring
that the latter is called with the right option (-string, -int or -real).


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK return if no errors occurred.  The result string is empty.
ERROR return if argument count is wrong or if seq_logFitsParRecord fails,
with a corresponding error message.


CAUTIONS
none


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>seq_logStringRecord OBS OBS "" NAME "this is my name"
seqWish>.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seq_logFitsParRecord(n), logFitsParRecord(3)




- - - - - -
Last change:  28/03/02-10:36
```

seq_logTclErrors(n)

```
NAME
seq_logTclErrors - send all output for stderr also to the error log


SYNOPSIS
seq_logTclErrors open|close


DESCRIPTION
seq_logTclErrors allows to send strings that normally go to exclusively
to stderr also to the CCS error log.  This feature is enabled by setting
the argument to "open", and disabled by setting it to "close".
Once active, errors logged will all have the same stack id, until this
logging is closed.

This command relies on a file-event.  Such events are only handled when
the Sequencer is idle, i.e. when it is not busy evaluating commands from
a script.  In other words, the output to the error log may not be visible
immediately, depending on the Sequencer's activity.


ENVIRONMENT
When this error logging is active, the global Tcl-variables OSE, EPR and
EPW are set and accessed.
The global Tcl-variable OSE refers to the original stderr file descriptor.
EPR and EPW are the reading resp writing end of a pipe created to divert
the messages sent to stderr.


RETURN VALUES
OK, with and empty result string if no errors
ERROR with corresponding message in the result string under the following
conditions:
- "called "seq_logTclErrors" with too many arguments" if there was more
  than 1 argument given to this command.
- "wrong arg: should be "open" or "close"" if argument is not correct
- "Tcl error logging already active" if open has been given before (i.e.
  the global Tcl-variable OSE exists)
- "Tcl error logging already disabled" if close has been given before
  (i.e. the global Tcl-variable OSE does not exist)


CAUTIONS
When this error logging is active, the Tcl-variables OSE, EPR and EPW are
set at a global level (by lack of static variables in Tcl).  They should
not be modified.

The overhead caused by sending these messages also to the CCS error system
is considerable (on top of adding possibly quite a few messages to the
log).  Its use should therefore be restricted to the bare minimum (e.g.
for debugging purposes).

Stderr must be open when Tcl error logging is started.


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
```

```
seqWish>seq_logTclErrors open
seqWish>blablabla
Error: invalid command name "blablabla"
seqWish>seq_logTclErrors close
seqWish>.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seq_errResetStack(n), seq_errAdd(n), seq_errCloseStack(n)




- - - - - -
Last change:  28/03/02-10:36
```

seq_msgCheck(n)

```
NAME
seq_msgCheck, seq_msgList - check and list commands with pending replies


SYNOPSIS
seq_msgCheck <cmdId> | {<envName> <procName> <command> }

seq_msgList ?{<envName> ?<procName> ?<command>??}?


DESCRIPTION
seq_msgCheck:
    checks if there are still replies pending on a command identified by
    its arguments.  A "1" is returned if replies are pending, "0" if not.

    <cmdId>     : unique command handle, as returned by seq_msgSendCommand
    <envName>   : environment name
    <procName>  : process name
    <command>   : command name

seq_msgList:
    returns a list with pairs, one pair per pending reply.  Each pair
    contains a cmdId and a sublist with originating environment name,
    process name and command name, describing the command for which a
    reply is pending.  The optional list allows to refine the reply-list
    progressively.

    <envName>   : environment name
    <procName>  : process name
    <command>   : command name


FILES
no files are accessed


ENVIRONMENT
no Tcl variables are read nor set


RETURN VALUES
OK if no errors occurred.  In this case the result string contains the
    result as indicated above.

ERROR if one of the following conditions is met (with the corresponding
error message here between quotes):
- seq_msgCheck:
  * "wrong # args: should be "seq_msgCheck <cmdId> | {<envName> <procName>
    <command>}""
  * "bad argument: "<argv[1]>" should be "<cmdId> | {<envName> <procName>
    <command>}""
  * error message left by Tcl_SplitList failing to parse the list
  * error message left by TclX_HandleXlate failing to parse the <cmdId>
- seq_msgList:                          (char *) NULL);
  * "wrong # args: should be "seq_msgList ?{<envName> ?<procName>
    ?<command>??}?""
  * "bad argument: "<argv[1]>" should be "?{<envName> ?<procName>
    ?<command>??}?""
  * error message left by Tcl_SplitList failing to parse the list
  * error message left by TclX_HandleXlate failing to parse the <cmdId>
```

CAUTIONS

The <cmdId> as returned by seq_msgSendCommand is a unique identifier, while <{<command> <envName> <procName>}> is not if several identical commands to a certain process have replies pending. Hence, it is strongly advised to use the former as argument for this command.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_msgSendCommand wte13 DCS init
cmd15
seqWish>...
seqWish>seq_msgCheck cmd15
1
seqWish>seq_msgList wte13
{cmd11 {wte13 ICS START}} {cmd15 {wte13 DCS INIT}}
seqWish>seq_msgRecvReply cmd15
CCD#87 initialized OK
seqWish>seq_msgCheck cmd15
0
seqWish>........
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqMsgSendCommand(3), seqMsgRecvReply(3)

- - - - - -
Last change: 28/03/02-10:36

seq_msgDispatch(n)

```
NAME
seq_msgDispatch, seq_msgDispatchBreak - enable asynchronous queue
                                    monitoring via file descriptor


SYNOPSIS
seq_msgDispatch ?<timeout> ?<nrMessages>??

seq_msgDispatchBreak


DESCRIPTION
seq_msgDispatch:
    the Sequencer command executed automatically whenever a message is
    available on the queue, and message reading has been enabled (see
    seq_css(n)).  As a registered Sequencer command it can be called just
    like any other command, although this is not its primary reason for
    existence.

    <timeout>    : timeout value in ms; default: 0 (wait forever).  Remark
                   that if there is no message getting on the queue within
                   this timeout, the Sequencer shell will not be responsive
                   for the duration of this timeout!
                   If the timeout is negative, a no-wait retrieval of
                   messages will be done, and no error will be reported if
                   the queue is empty.
    <nrMessages> : number of messages to get; default: 1

    The standard CCS message-types can be retrieved this way, including
    obituaries (provided this shell's CCS process-name has been entered in
    the RtapEnvTable, with a proper value in the 'Care about Terminations'
    field; remark that processes doing a ccsInit/ccsExit will generate
    obituaries by default).
    When an obituary is received, the script contained in the global
    variable seq_obiScript will be executed, after the following
    substitutions took place:
    - %D: 1 if the deceased process registered for debug, 0 otherwise
    - %E: environment name of the deceased process/environment
    - %P: deceased process name or {} if the environment deceased
    - %R: 1 if the deceased process will be automatically restarted, 0
          otherwise.
    - %S: exit status of the terminating process (integer value)
    - %%: a single % character
    Internal tables related to the deceased process (like the ones kept for
    commands with pending replies) will be cleaned upon reception of an
    obituary message. This internal housekeeping takes place even if
    seq_obiScript is empty, and is actually done before evaluating
    $seq_obiScript.

    When seq_msgDispatch gets a command from the message queue it will
    execute it.  Presently the following commands are recognized (on top of
    the special commands BREAK, KILL and PING predefined for all CCS
    applications):
    - EVENT : the message body must contain a legal Sequencer script which
            will be evaluated.  Similar to the SCRIPT command (see below),
            with the difference that no replies are generated: the sender
            of the EVENT command will not receive and should not expect any
            reply on this command. This command is useful in an environment
            where some process(es) need(s) to be informed of an event,
            while it is considered too expensive to create/access a
            database branch just for this purpose.
```

```
                If this command was sent via seq_msgSendCommand, the latter
                needs to have the -noreply option specified.
        - SCRIPT : the message body must contain a legal Sequencer script which
                will be evaluated.  The result of this evaluation (i.e. the
                return string) will be sent as the second reply to this
                command - an immediate, empty reply is given before the
                evaluation starts.  If in this script intermediate replies
                have to be sent, the so-called script-command-handle will be
                needed (as this allows to retrieve the originator of the
                command).  This handle can be obtained with a %-substitution,
                performed on the script before evaluation (equivalent to the
                seq_evt substitutions):
                - %S : script-command-handle; it is needed by the
                       seq_msgSendReply command, as it allows to identify the
                       process which originated the SCRIPT command..
                - %% : a single % character
                If the result of the script-evaluation is a string of a size
                which does not fit into a single reply, seq_msgDispatch will
                automatically chop up this string into multiple fragments,
                send each fragment (in the right order) as a reply, whereby
                only the last fragment will have the 'lastReply' flag set.
                It will be up to the sender of the SCRIPT command to re-assemble
                these replies by concatenating these individual pieces.

        All other commands will be logged as errors, and an error message will
        be sent back as well to the originator of the command.

        seq_msgDispatch does also react on events.  Whenever an event message
        comes in, the script given at the time this event was defined (see
        seq_evtAttach(n) and seq_msgRecvReply(n)) will be executed.  The
        evaluation of the script is at global level.
        Whenever the substitution on the original script or the evaluation of
        the resulting script yields an error, the corresponding event will be
        disabled, and the error will be logged and also signalled via
        Tcl_BackgroundError(3).

        Similar for alarms:  whenever an alarm message comes in, the script
        given at the time this event was defined (see seq_alrmAttach(n)) will
        be executed.  The evaluation of the script is at global level.
        Same remark about script substitution/evaluation errors as for event
        messages.

        The setting up of the queue monitoring is done via a call to
        seqMonitorQ(3).  Also the suspension of this service can be requested
        to the same routine.

        Whenever the timeout (in msec) or the nrMessages to read leads to an
        unacceptable blocking of other tasks, the read-and-dispatch loop can be
        broken with the seq_msgDispatchBreak command.  Timeout defaults to 0
        (wait forever), while nrMessages defaults to 1.  The timeout is the
        time to wait on a msgRecvMsg(3) call for a single message, i.e. it is
        not the cumulative timeout, nor does it take the time to execute the
        command (contained in the message) into account.

    seq_msgDispatchBreak:
        breaks the execution of seq_msgDispatch


    ENVIRONMENT
    seq_obiScript is a global Tcl variable containing the script to be executed
    when an obituary message is received. This script can contain some variables
    (single characters preceded by a %-sign) giving information about the
    obituary itself; they are substituted just before evaluation of the script.
```

RETURN VALUES

seq_msgDispatchBreak always returns an OK; the remainder of this sections is for seq_msgDispatch only.

OK if no errors occurred; the reply contains the number of messages effectively dispatched. Remark that this may deviate from <nrMessages> depending on the setting of <timeout> and the intermediate reception of a seq_msgDispatchBreak command.

ERROR if one of the following conditions is met (corresponding error message is here between quotes):

- "wrong # args: should be "seq_msgDispatch ?<timeout> ?<nrMessages>??"" if the command contains more than 2 arguments
- "bad argument: "<argv>" should be an integer" if either <timeout> or <nrMessages> have illegal values
- "bad argument: "<argv>" should be positive" if <nrMessages> is a negative integer
- "could not get msg queue file descriptor" if seqGetMsgFd(3) failed.
- "timed out after <xx> messages"; can obviously only occur if <timeout> is greater than 0.

Remark that most of these messages cannot appear when the seq_msgDispatch is called via the "fileevent" scheme. In this case we are 100 % sure the arguments are set to legal values and there is a message on the queue. Also internal messages, which cannot be retrieved by the application but do cause a fileevent, do not provoke any errors.

The causes of these errors are also logged via the standard CCS error system and logging.

CAUTIONS

Incoming commands cannot be longer than ~8K (i.e. the limit of a single CCS message).

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_msgDispatch 1000 1
1
seqWish>seq_msgDispatch 100000 5
5
seqWish>....
.....
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seq_ccs(n), seq_evtAttach(n), seq_evtParseMsg(3), seqMsgDispatch(3), seqMonitorQ(3), RtapEnvTable(4)

- - - - - -
Last change: 28/03/02-10:36

seq_msgDispatchBreak(n)

See seq_msgDispatch(n).

- - - - - -

Last change: 28/03/02-10:36

seq_msgFlush(n)

```
NAME
seq_msgFlush - flush pending replies


SYNOPSIS
seq_msgFlush ?<option>? <filter>


DESCRIPTION
seq_msgFlush waits a certain time for all replies to flush them as they
come in, or optionally flushes replies that are already available.

<option> can be either:
    * -nowait : flags not to wait, i.e. flush only replies which are
                available at the time the command is given
    * <nnn> : an integer giving the timeout in ms to wait for replies
    If no <option> is specified, the wait will be forever.

<filter> is a list with at least one element; it is either:
    * the environment, and optionally, the process name for which
      we want to flush the replies
    * the word "all" to indicate replies to all commands



ENVIRONMENT
No Tcl variables are accessed.


RETURN VALUES
OK, with and empty result string if no errors
ERROR with corresponding message in the result string under the following
conditions:
- "called "seq_msgFlush" with too many arguments" if there were more
  than 2 arguments given to this command.
- "wrong filter: should be "{<envName> ?<procName>?}"" if <filter> is wrong
- "wrong option: should be "-nowait" or "<nnn>"" if <option> is not correct
- any error message returned by seq_msgRecvReply


CAUTIONS
The timeout given is applied to all replies separately, i.e. the total,
cumulative timeout may be much larger than given by this number


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>seq_msgList
{cmd5 {wte13 OS INIT}} {cmd9 {lte18 ICS RESET}} {cmd27 {wte13 DCS INIT}}
seqWish>seq_msgFlush wte13
seqWish>seq_msgList
{cmd9 {lte18 ICS RESET }}
seqWish>.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
```

```
seq_msgRecvReply(n)




- - - - - -
Last change:  28/03/02-10:36
```

seq_msgList(n)

```
See  seq_msgCheck(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_msgRecvReply(n)

```
NAME
seq_msgRecvReply - receive a reply on a CCS command sent before


SYNOPSIS
seq_msgRecvReply ?<options>? <cmdId> <errorNumber> ?<lastReply>?


DESCRIPTION
seq_msgRecvReply waits for a single or all replies on a command identified
by its first argument, or for the specified timeout - whichever comes
first.

<options>: any of the following options, in any order (if contradicting
    options appear, the last one will prevail):
    * -all    : flags to get all replies in, instead of only one, and
                return all replies separated by a newline (in absence of
                option "-bin"); empty replies do not get a newline
                appended, nor is there a newline appended to each reply
                if the -nonewline option is specified.
    * -ascii  : flags reply is supposed to contain ASCII data; the CDT's
                data for the reply's format is not checked nor used.  This
                is slightly more efficient if you know in advance that the
                reply is ASCII (and/or that the CDT is not available)
    * -bin    : flags data should not be returned in result string
                (e.g. when reply contains unformatted binary data).
    * -fbin   : flags data should be formatted according to the command's
                entry in the CDT (DISPLAY_FORMAT field). This works fine
                for formatted binary and ASCII replies, although for the
                latter it is more efficient to use the -ascii option (so no
                formatting needs to be attempted). Unformatted binary
                replies will result in an empty string (although again, it
                is more efficient to do that via the -bin option).  If the
                reply does not have an entry in the CDT, it will be
                considered as ASCII (same remark again).
    * -last   : flags to wait on the last reply, and and return the data of
                the last reply only (if option "-bin" is not set)
    * -<nnn>  : any integer, giving the timeout in ms; absolute value is
                taken
    * -nonewline: indicate that the concatenation of multiple replies
                (see -all option) should be without additional newlines.
                Specifying this option in the absence of "-all" does not
                have any effect.
    * -nowait : flags no wait, i.e. first the internal queue is checked,
                and if no reply is found a call to msgRecvMsg() is placed
                with the appropriate filter and msgNO_WAIT flag
   The option -ascii is enabled by default if seq_ccsCmdCheck was set to
   "CHECK_CMD" when the command was sent; otherwise the -fbin option is
   active.  These defaults may be overridden by command line options.

<cmdId>: as returned by seq_msgSendCommand

<errorNumber>: the name of a Tcl variable where to store the error-number
    of the error-message structure returned as part of the reply.  The
    CCS convention is that "0" signals no error.  This variable can only
    be set after the message was received OK (i.e. one should test first
    that the reply-protocol did not fail, then look at <errorNumber>).
    When the "errorNumber" variable indicates the reception of an
    error message, this error-stack can be displayed with seq_errDisplay
    (until the next error comes in).
```

```
<lastReply>: the name of a Tcl boolean variable where to store the
    "lastReply" flag; will contain "0" if there are more replies, "1" if
    this reply is the last.  This variable will not be touched untill the
    reply(-ies) came in successfully.

Remark that imbedded applications (which call seqInitInterp()) can have
an event-loop out of the sequencer's control, i.e. they can call ccsInit()
instead of eval-ing seq_ccsInit (which would activate the monitoring of
the message queue within the sequencer's event-loop). In these cases,
seq_msgRecvReply will by default issue a blocking msgRecvMsg() call with
infinite timeout, as it cannot count on the sequencer's monitoring of the
message queue.


FILES
no files are accessed


ENVIRONMENT
<errornumber> and <lastReply>: see above

The value of the seq_ccsCmdCheck variable at the time the command was sent
is used to flag the (expected) presence of the CDT of the process the reply
comes from.  It influences the default behaviour of seq_msgRecvReply
(-ascii or -fbin).


RETURN VALUES
OK if the reception of the reply caused no errors.  In this case the
result string contains either the reply or error message received.  An
eventual error will be returned independent of the setting of the -bin
option.
Remark that this OK return does not necessarily mean that the originator
of the reply signalled all is OK - cf the <errorNumber> variable.

ERROR if one of the following conditions is met (with the corresponding
error message here between quotes):
  * 'wrong # args: should be "seq_msgRecvReply ?-all? ?-ascii? ?-bin?
    ?-fbin? ?-last? ?-nnn? ?-nonewline? ?-nowait? <cmdId> <errorNumber>
    ?<lastReply>? "' if there were not at least two additional arguments
    (on top of eventual options)
  * 'invalid cmd handle: "<cmdId>"'
  * 'bad option: "<optionX>" should be -all, -ascii, -bin, -fbin, -last,
    -nowait, -nonewline or a numeric value' if one of the options is wrong
  * 'invalid cmd handle "<cmdId>" (no replies pending on it)'
  * 'failed to allocate memory to format binary reply' if there seems
    to be no memory available for the temporary formatting buffer; this
    allocation is done the first time formatting of a reply is attempted.
  * 'reply on <cmdId> (command <cmd> to <env>/<proc>) could not be
    formatted' if the formatting failed (most likely there is something
    wrong with the CDT - check the error stack and CDT).
  * 'reply timed out' if no reply came in within the specified timeout
  * 'message queue empty' if -nowait was specified and no reply was in
    yet
  * 'could not receive message' if msgRecvMsg returned an error
  * 'could not parse reply'
  * 'received reply on "<command>" with wrong commandId (<commandId>)'
    when the replier either did not set the orgCommandId properly, or
    sends more replies after the lastReply flag was set
  * 'no memory available to store replies' if in the handling of incoming
    replies a malloc failed
  * error message left by Tcl_SetVar when trying to set the variable
    <lastReply>; remark that when this error occurs, the reply was
```

received properly.

CAUTIONS

The identification of the command that caused a reply depends on the the commandId parameter sent together with the command. It is therefore absolutely essential that all applications receiving Sequencer commands via the CCS message system, return replies that contain the CommandId of the originating command.

When a reply comes in with the lastReply flag set, the Sequencer shells will remove all information pertinent to this command after dealing with this reply. If further replies would come in, they will be properly logged as errors, and should be interpreted as an indication of a buggy replying application.

For formatted binary replies, the formatted string of a single reply is limited to 16 KBytes. There is however no checking for overflow possible (due to how the CCS procedure cmdFormatReply is implemented), and if the DISPLAY_FORMAT entry in the corresponding CDT leads to larger formatted strings, a memory corruption will occur.

To be able to format replies, the CDT must be loaded into cmdManager. See cmdManager(1) and cmdSetup(1).

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_msgSendCommand wte13 DCS init
cmd145
seqWish>.....
seqWish>seq_msgRecvReply cmd145 errNr lr
CCD#87 having problems, trying again...
seqWish>echo $lr
0
seqWish>seq_msgRecvReply cmd145 errNr lr
CCD#87 initialization failed
seqWish>echo $lr
1
seqWish>echo $errNr
15
seqWish>seq_msgSendCommand wte13 DCS init
cmd187
seqWish>seq_msgRecvReply cmd187 errNr lr -last
CCD#87 initialized OK
seqWish>echo $lr
1
seqWish>echo $errNr
0
seqWish>........
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqMsgRecvReply(3), seq_msgSendCommand(n), msgRecvMsg(3), msgParseMsg(3)

```
- - - - - -
Last change:  28/03/02-10:36
```

seq_msgSendCommand(n)

NAME

seq_msgSendCommand - send a CCS message

SYNOPSIS

```
seq_msgSendCommand ?-noreply? ?-check? ?-nocheck? \
                   <envName> <procName> <command> ?<args>?
```

DESCRIPTION

```
-noreply   : flag that this command is of an event type, i.e. there will be
             no replies to be sent back
-check     : flag to request command and parameter checking for this
             particular command, overriding the global setting of the
             seq_ccsCmdCheck Tcl-variable (see below).
-nocheck   : flag to turn off command and parameter checking for this
             particular command, overriding the global setting of the
             seq_ccsCmdCheck Tcl-variable (see below).
<envName> : name of the environment where the command has to be sent to
<procName>: name of the recipient process
<command> : name of the command to send to the environment/process
<args>     : the arguments to pass along with <command> as the message body.
    This can spread multiple words, and the Sequencer will concatenate
    them into a single message body (string), separating neighbouring
    arguments by a single space (similar to Tcl's join command).  Remark
    however that in this case the Tcl parser and the Sequencer need to do
    some work on <args> which can be avoided: it is more efficient to have
    <args> grouped as a single word, e.g. between quotes, or, if there are
    no substitutions required, between curly braces.
    The joining of the <args> happens of course after applying the standard
    Tcl substitution- and grouping-rules.  When arguments containing
    quotes are involved, unexperienced Tcl users will often consider this
    an odd behaviour. Again, this situation can be avoided by grouping
    <args> into a single Tcl-word (see EXAMPLES section).
```

In the absence of -noreply, and when send_msgSendCommand is successful, it returns the next available command handle. This handle points to a.o. the msgCMDID parameter sent to the destination. The latter msgCMDID is derived from the handle name (it is the numeric part of it), and it can be used to construct a filter for receiving replies on this command. It is therefore imperative that the recipients of such messages return the original command identifier in the reply.

If -noreply is given, the command will be sent with a msgCMDID of value 0, and the return will be empty, as there are supposed to be no replies coming back. This option is useful in an environment where some well-known process(es) need(s) to be informed of an event, while it is considered too expensive to create/access a database branch just for this purpose. This option is functionally equivalent to sending a normal command and then receiving and discarding replies (e.g. via the seq_evtAttach mechanism), but it is of course a lot more efficient.
If the recipient process is a Sequencer shell, the <command> needs to be EVENT instead of SCRIPT, as this disables the return of replies by the recipient.

The checking of commands is in the absence of the -cmdCheck and -noCheck options done according to the value of the global Tcl-variable seq_ccsCmdCheck.

FILES

no files are accessed

ENVIRONMENT

seq_ccsCmdCheck is a global Tcl_variable whose value determines whether the CCS message system will perform command checking based on the CDTs. It must be set to either NO_CHECK or CHECK_CMD. The default is NO_CHECK. Its application can be overridden by the options -check and -nocheck. See also the description of the argument flag in the manpage of msgSendCommand(3).

RETURN VALUES

OK if no errors occurred. In this case the result string is either empty if -noreply was specified, or otherwise contains the next available command handle, as mentioned above.

ERROR if one of the following conditions is met (with the corresponding error message here in bold-face):

- wrong # args: should be "seq_msgSendCommand ?-noreply? ?-check? ?-nocheck? <envName> <procName> <command> ?<args>?" if seq_msgSendCommand is not given with at least 3 additional arguments
- wrong option: "-xyz" should be "-noreply", "-check" or "-nocheck" if an unknown option -xyz was specified
- could not send command "<command> <args>" to <envName>/<procName> if msgSendCommand(3) returned an error

CAUTIONS

<command> will be converted to UPPERCASE, as required by CCS! As a consequence, the Sequencer shells cannot deal with command-synonyms containing lowercase characters.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish># The trivial case: 3 parameters without quotes and spaces
seqWish>seq_msgSendCommand -check wte13 ICS TEST {par1 2 par3}
cmd14
seqWish># The same result, easy to grasp, but some more arbeit for seqWish
seqWish>seq_msgSendCommand -check wte13 ICS TEST par1 2 par3
cmd15
seqWish># The easy way to pass a parameter including quotes
seqWish>seq_msgSendCommand wte13 ICS TEST {1 2 "3rd parameter with quotes"}
cmd16
seqWish># The same result, but novices may have to think a bit about this
seqWish>seq_msgSendCommand wte13 ICS TEST 1 2 {"3rd parameter with quotes"}
cmd17
seqWish># Here the third parameter is a string which does not include
seqWish># quotes; as it contains spaces, the whole is after concatenation
seqWish># with the first two arguments equivalent to a string with 6
seqWish># individual parameters
seqWish>seq_msgSendCommand wte13 ICS TEST 1 2 "3rd 4th 5th 6th"
cmd18
seqWish>........
```

SEE ALSO

Tcl and the Tk toolkit, John K. Ousterhout, ISBN 0-201-6337-X

```
seqMsgSendCommand(3), seq_msgRecvReply(n), msgSendCommand(3),



- - - - - -
Last change:  28/03/02-10:36
```

seq_msgSendReply(n)

```
NAME
seq_msgSendReply - send a CCS reply message


SYNOPSIS
seq_msgSendReply -current|<scriptId> <errorNr> ?<lastReply>? <reply>


DESCRIPTION
seq_msgSendReply allows to send intermediate replies to the originator of
the SCRIPT command (see seq_msgDispatch(n)).  As such, the only place where
it can appear is within the script received with this SCRIPT command, or
within a procedure called by this script.

-current: a flag indicating that the scriptId of the current SCRIPT command
          should be used instead of an explicitly specified <scriptId>. This
          provides an alternative to the %S substitution (see below).
          In case various SCRIPTS commands are nested (possible if a script
          being evaluated includes e.g. an "update" command), the handle
          used is always the one of the current context.

<scriptId>: script-command handle as substituted for a %S by seq_msgDispatch
            at the time it received the SCRIPT command; this handle allows
            to identify the process which originated the SCRIPT command.

<errorNr>:  number of the error; 0 means no error

<lastReply>: a Tcl boolean variable, to flag if this is the last reply.
             Its default value is 0. If <errorNr> is not 0, the value
             specified for <lastReply> is irrelevant, i.e. an error-reply
             is by definition the last reply.

<reply>:    string containing reply; if <errorNr> is 0, this will be in the
            message buffer at the receiving side; otherwise, this string
            will be in the ccsERROR structure passed across (which means
            this is then also the last reply).
            If <reply> is of a size which does not fit into a single reply,
            seq_msgSendReply will automatically chop up this string into
            multiple fragments, send each fragment (in the right order) as
            a reply, whereby only the last fragment will have the
            'lastReply' flag set, provided <lastReply> is true. It is up
            to receiver of these replies to re-assemble them by
            concatenating these individual pieces.


FILES
no files are accessed


RETURN VALUES
OK if no errors occurred.  In this case the result string is empty,
except if the -current option was given; in the latter case, the
<scriptId> is returned.

ERROR if one of the following conditions is met (with the corresponding
error message here between quotes):
  * 'wrong # args: should be "seq_msgSendReply -current|<scriptId>
    <errorNr> ?<lastReply>? <reply>"' if seq_msgSendReply is not given with
    the right amount of arguments
  * 'not evaluating a SCRIPT command' when the command was used with the
    -current option, while not evaluating a script that came with a SCRIPT
```

command.

* 'no replies to be sent for <scriptId>' when the <scriptId> handle does not or no longer exist (e.g. the last reply was sent before).
* 'could not send reply "<reply>" to <envName>/<procName>' if msgSendReply returned an error
* 'could not send reply "<reply>", nor get procName from msgPROCESSID' if msgSendReply returned an error, and also ccsGetProcName failed.

CAUTIONS

The <errorNr> (if set) will be in the top of the error stack structure wich is sent as a reply, and can be retrieved as such by the receiving side. This means that the generic error mnemonic seqERR_CMD_SCRIPT used for these cases does not correspond to a fixed error number - its number gets overwritten with <errorNr>.

EXAMPLES

(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
....
seqWish>seq_msgSendReply script0 0 "I am still doing OK"
seqWish>........
seqWish>seq_msgSendReply script0 5 "I have found an error: $msg"
seqWish>........
```

SEE ALSO

"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
seqMsgDispatch(n), seq_msgRecvReply(n), msgSendReply(3),

- - - - - -

Last change: 28/03/02-10:36

seq_oslxCmd(n)

```
NAME
seq_oslxCmd - process an oslx-command


SYNOPSIS
seq_oslxCmd <command> <parameter> ?<instanceId>? ?<arguments>?


DESCRIPTION
seq_oslxCmd will invoke oslxSHELL::CmdParser, with the following
arguments:
<command>   : name of a command supported by oslxServer
<parameter> : parameter-name going with this command (see the CDT of
              oslxServer).
<instanceId>: instance Id of the command; must be unique.  Remark that not
              all commands require this parameter
<arguments> : additional arguments of this <command> to be passed to
              oslsServer; how many of these arguments there are, depends
              on the specific <command>.


ENVIRONMENT
no environment nor Tcl variables are read nor set


RETURN VALUES
OK return if no errors occurred.  Also, the reply-buffer of this <command>
    will be returned in the result string.
ERROR return if one of the following conditions is met (with corresponding
error message here between quotes):
  - 'wrong # args : should be "seq_oslxCmd <command> <parameter>
    ?<instanceId>? ?<arguments>?"': the corresponding Sequencer command
    was given with too few arguments;
  - 'could not parse "<command> <parameter> ..." (oslxERR_...)' if oslx
    returned an error (the error message from oslx is included);


CAUTIONS
Remark that although the generic command syntax rules allows several
parameters and their corresponding values to be combined in a single
command, oslxServer does not do so.  So <parameter> is a *single*
parameter of <command>.

Some of the oslxServer commands affect environment variables. In order
to keep the Tcl-array "env" aligned with the process' environment, it
is therefore imperative that the seq_oslxCmd command is given within a
shell which takes care of this aspect, like seqSh or seqWish.


EXAMPLES
(For clarity's sake, the example is for an interactive session with
input typed by the user coming after the "seqWish>" prompt, and all
replies on a new line)

.....
seqWish>
seqWish>.....


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
```

```
oslxServer(1)




- - - - - -
Last change:  28/03/02-10:36
```

seq_redirect(n)

NAME
seq_redirect - redirect fileId to a disk file

SYNOPSIS
seq_redirect <fileId> ?<fileName> <checkPeriod> <maxSize>?

DESCRIPTION
When all 4 arguments of this command are specified, seq_redirect will redirect <fileId> to <fileName>. This disk file is monitored each <checkPeriod> seconds and whenever its size exceeds <maxSize> bytes only the last part (10% of <maxSize>) will be kept.

If only the first argument is given, this logging to a file for <fileId> is stopped.

This command is particularly useful to redirect e.g. stderr to a log-file, as debugging info is always written to stderr (see seq_debug array)

ENVIRONMENT
no environment nor Tcl variables are read nor set

RETURN VALUES
returns OK, with an empty result string if all is OK.

returns ERROR with corresponding message in the result string if a file access error occurred, or if attempting to stop redirection for a fileId which is not being redirected.

EXAMPLES
(For clarity's sake, the example is for an interactive session with input typed by the user coming after the "seqWish>" prompt, and all replies on a new line)

```
.....
seqWish>seq_redirect stderr /tmp/myLogFile 3600 1000000
seqWish>
.....
seqWish>seq_redirect stderr
seqWish>
```

SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X

- - - - - -
Last change: 28/03/02-10:36

seq_relToAbsPath(n)

```
NAME
seq_relToAbsPath, seq_findFile, seq_waitTillActive, seq_waitTillDead,
seq_isoTime, seq_fitsDate, seq_timeOfDay, seq_isoTimeToClock -
    a collection of Sequencer procedures


SYNOPSIS
seq_relToAbsPath <pathName>

seq_findFile <relPath>

seq_waitTillActive <env> <proc> <timeout> \
                  ?<period> ?<runstring> ?<preferExec>???

seq_waitTillDead <env> <proc> <timeout> ?<period>? ?<exitCmd>?

seq_isoTime

seq_fitsDate

seq_timeOfDay

seq_isoTimeString2Clock <timeString>


DESCRIPTION
seq_relToAbsPath will return the absolute pathname corresponding to a
   relative <pathName>; works for files as well as directories

seq_findFile will return the absolute path of <relPath>, after looking
   for it in the following order:
       1. the current working directory, i.e. ./<relPath>
       2. ../<relPath>
       3. $INTROOT/<relPath>
       4. $VLTROOT/<relPath>
   If <relPath> is not found, an error is returned ('<relPath> not found')

seq_waitTillActive will wait untill a certain process is active, i.e. until
   it can receive CCS messages; this is tested by sending periodically PING
   commands with the CCS message system
   <env>    : environment where the process will run
   <proc>   : name under which the process will register itself with CCS
   <timeout>: how long to wait (in ms) before the seq_waitTillActive
              returns an error
   <period> : period (in ms) for sending PING commands; default: 100 ms.
              This determines the resolution of <timeout>
   <runstring>: If not empty, seq_waitTillActive will first of all schedule
              the process in the environment <env>.  If this <env> is the
              local environment, "exec" can be used for this scheduling -
              see <preferExec> - thereby setting stdout of this new process
              the same as the script's stdout; otherwise, the msgSchedule(1)
              utility is used (and there will be no stdout messages).  In
              both cases, the <runstring> is passed to the scheduling
              utility.
   <preferExec>: a boolean variable; if set to 1 and <env> is the local
              environment, use exec(n) for the scheduling instead of
              msgSchedule(1) (see above). Default: use msgSchedule(1).
   There are no errors logged for the failed attempts to send a PING
   command, nor for the failing of seq_waitTillActive itself.
   If the scheduling is done with exec(n), the Unix process-id will be
   returned if successful.
```

```
seq_waitTillDead will wait untill a certain process is inactive, i.e. until
   it no longer responds to CCS PING messages
   <env>    : environment where the process is running
   <proc>   : name under which the process is registered with CCS
   <timeout>: how long to wait (in ms) before the seq_waitTillDead
              returns an error; remark that this timer starts *after* the
              reply to <exitCmd> came in.
   <period> : period (in ms) for sending PING commands; default: 100 ms.
              This determins the resolution of <timeout>
   <exitCmd>: If not empty, seq_waitTilDeadlActive will first of all send
              the command <exitCmd> to <proc>, using seq_msgSendCommand,
              before starting to test if it is still active. If this
              parameter is given, seq_waitTillDead wait for a reply on this
              command and wait for an additional 2 seconds before polling
              with the PING command.

   If the <exitCmd> argument is not given, the script calling this command
   should make sure (e.g. using a delay) that the process had a reasonable
   chance to terminate, before calling seq_waitTillDead. Otherwise the PING
   commands may still get on the queue of the terminating process while it
   is exiting and therefore it would not reply. Although seq_waitTillDead
   would finally timeout after <timeout> ms, and consider the process for
   that reason dead, this "delay" in coming to that conclusion could
   surprise the user.

   There are no errors logged for the failed attempt to send a PING command,
   nor for the failing of seq_waitTillDead itself. If seq_waitTillDead
   cannot communicate with <env>/<proc>, it is assumed that the process
   is dead already (or that <env> is no longer active), and a OK-return
   will be taken.

seq_isoTime will return the current UTC time in ISO 8601 format (i.e. in the
   format 'yyyy-mm-ddThh:mm:ss').

seq_fitsDate will return the current UTC date in the format 'yyyy-mm-dd'

seq_timeOfDay will return the current UTC time-of-the-day in the format
   'hh:mm:ss'.

seq_isoTimeToClock will convert the UTC time <timeString> (ISO-format) to
   the internal clock value (integer, timezone adjusted) corresponding to
   this UTC time.  This is useful if one wants to make elapsed time
   calculations starting from ISO-format time-strings, or to convert an
   ISO <timeString> to another non-ISO format (e.g. "clock format
   [seq_isoTimeToClock <timeString>]" will return the time in the format
   as the standard Unix "date" command).


RETURN VALUES
as any Tcl procedure; OK returns are always empty, except for
seq_waitTillActive when "exec" is used; in this case, the Unix process-id
of the scheduled process is returned.

In case of an error, the following self-explanatory error messages can be
returned:
seq_findFile:
    '<relPath> not found in standard search path''
seq_waitTillActive:
    'process <proc> (env <env>) not active after <timeout> ms'
seq_waitTillDead:
    'process <proc> (env <env>) still active after <timeout> ms'
```

CAUTIONS

If seq_waitTillDead cannot communicate with <env>/<proc>, it is assumed that the process is dead already, and a OK-return will be made. This behaviour is desired in most cases. Beware however that this OK-return will also occur when e.g. <env> has a wrong value.

On top of that, timeouts for replies on accepted PING commands (both for seq_waitTillActive and seq_waitTillDead) result in the automatic deletion of the corresponding command-handle.

All time-related procedures require that the TimeZone is set to the proper value, as time-values are converted to/from UTC based on this TZ info.

SEE ALSO

clock(n), msgSchedule(1)

- - - - - -

Last change: 28/03/02-10:36

seq_scanConfig(n)

```
NAME
seq_scanConfig - configure a db entry to be acquired by the scan system


SYNOPSIS
seq_scanConfig arguments


DESCRIPTION
see manpage for scanConfig(1).


ENVIRONMENT
No Tcl variables are accessed.


RETURN VALUES
OK, with a result string containing all what scanConfig(1) sent to stdout,
if it terminated OK (exit(0)).
ERROR with corresponding message in the result string if scanCOnfig(1)
terminated abnormally (exit(1)).


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
scanConfig(1)




- - - - - -
Last change:  28/03/02-10:36
```

seq_timeOfDay(n)

```
See  seq_relToAbsPath(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_waitTillActive(n)

```
See  seq_relToAbsPath(n).


- - - - - -
Last change:  28/03/02-10:36
```

seq_waitTillDead(n)

See seq_relToAbsPath(n).

- - - - - -

Last change: 28/03/02-10:36

seqCon(n)

```
NAME
seqCon - start up tkcon within the seqWish interpreter


SYNOPSIS
source seqCon.tcl


DESCRIPTION
seqCon is a script which will invoke tkcon, and "customize" it to the
seqWish style.
The script will however return without action if the seqWish runstring
includes -noTkCon, or if this is a non-interactive shell.


ENVIRONMENT
The argv array is checked for the presence of the switch "-noTkCon",
in which case tkcon will not start up.


SEE ALSO
"Tcl and the Tk toolkit", John K. Ousterhout, ISBN 0-201-6337-X
tkcon documentation (http://www.cs.uoregon.edu/research/tcl/script/tkcon/)



- - - - - -
Last change:  28/03/02-10:36
```

5 PUBLIC INTERFACE FILE

```
#ifndef SEQ_H
#define SEQ_H
/*******************************************************************************
* E.S.O. - VLT project
#
# "@(#) $Id: seq.h,v 2.69 2002/02/15 14:04:41 vltsccm Exp $"
*
* seq.h
*
* who       when      what
* --------  --------  ----------------------------------------------
* eallaert  30/03/94  created
* eallaert  28/07/95  differentiation of seqLONGNAME according to use of CCS
* eallaert  11/10/95  fixed various problems for compilation without CCS
* eallaert  03/05/96  Seq_Init name compatible with "load" in Tcl 7.5
*/

/************************************************************************
*  This is the public interface file for the Sequencer extensions to Tcl/Tk
*
*  Remark that if you use it, the following compilation flags should be set:
*  - CCS_FULL if the compilation is with full CCS
*  - CCS_LIGHT if the compilation is with CCS-light
*  - neither of the above if the compilation is without CCS
*
*----------------------------------------------------------------------
*/
/*
 * Other header files
 */
#if defined(CCS_FULL) || defined(CCS_LIGHT)
#include "seqErrors.h"          /* error interface file */
#else
#include "seqErrorsNoCcs.h"     /* error interface file */
#endif

#include <tclExtend.h>          /* interface file for Tcl/TclX */

#if defined(CCS_FULL) || defined(CCS_LIGHT)
#include <ccs.h>                /* declares the vlt* and ccs* datatypes */
#endif  /* CCS_FULL || CCS_LIGHT */
/*
 * Definition of constants
 */
#define seqVERSION "$Revision: 2.69 $"  /* string containing revision code */
                                /* "tclAppVersion" points to digits only */
                                /* "infox appversion" returns digits only */

#define seqNAME "sequencer"     /* default name stored in tclAppName */

/* string returned by the command "infox applongname" for all CCS-cases
 * (this string is pointed to by tclAppLongName)
 */
#if defined(CCS_FULL)
#define seqLONGNAME "The VLT Sequencer with full CCS"
#define seqCCSTYPE "full"
#elif defined(CCS_LIGHT)
#define seqLONGNAME "The VLT Sequencer with CCS-light"
#define seqCCSTYPE "light"
#else
```

```
#define seqLONGNAME "The VLT Sequencer without CCS"
#define seqCCSTYPE "none"
#endif

#define tclPRECISION "tcl_precision"    /* Tcl-var linked to seqPrecision */
#define seqMODULEID "seq_moduleId"      /* Tcl-var linked to seqModuleId */
#define seqERRLOGGING "seq_errLogging"  /* Tcl-var linked to seqErrLogging */
#define seqERRREPLYMERGING "seq_errReplyMerging"/* linked to seqErrReplyMerging */
#define seqCMDCHECK "seq_ccsCmdCheck"   /* Tcl-var linked to seqCmdCheck */
#define seqOBISCRIPT "seq_obiScript"    /* Tcl-var linked to seqObiScript */

#define seqREPLYTIMEOUT "reply timed out"       /* return string value for a */
                                        /* timedout reply */

#define seqDEFAULTID "seq"              /* default module Id - cf errAdd() */
                                        /* package name is derived from this */
                                        /* (1st character is upshifted) */
#define seqDEFAULTCHECK "NO_CHECK"      /* def check - cf msgSendCommand() */

#define seqCMD_HANDLE_BASE "cmd"        /* basis for command handle name */
#define seqEVT_HANDLE_BASE "event"      /* basis for event handle name */
#define seqSCR_HANDLE_BASE "script"     /* basis for script handle name */
#define seqLIST_HANDLE_BASE "list"      /* basis for db-list handle name */
#define seqALRM_HANDLE_BASE "alarm"     /* basis for alarm handle name */


/*
 * Global variables-  see also seqInit.c, seqMsgDispatch
 */
extern unsigned int     seqPrecision;   /* C-variable linked to tclPRECISION */
                                        /* type must correspond to int */

extern char     *seqModuleId;           /* C-variable linked to seqMODULEID */
                                        /* see a.o. errAdd(); max 6 chars!!! */

extern int      seqErrLogMethod;        /* C-var reflecting seqERRLOGGING */
                                        /* 0=off, 1=auto, 2=manual */

extern int      seqErrReplyMergeMethod; /* C-var corr. to seqERRREPLYMERGING */
                                        /* 0=off, 1=internal, 2=local */

extern char     *numericVersion;        /* string containing version */

#if defined(CCS_FULL) || defined(CCS_LIGHT)

extern ccsERROR         lastError;      /* CCS error at last errCloseStack */

extern char             *seqObiScript;  /* C-var linked 2 string seqOBISCRIPT */
                                        /* cf obituaries in seqMsgDispatch */
extern ccsENVNAME       myEnvName;      /* local environment name */
extern ccsPROCNUM       myProcNum;      /* Sequencer process number */
extern ccsPROCNAME      myProcName;     /* the CCS name of this process */

extern vltLOGICAL       cmdsAllowed;    /* reflects whether Sequencer event */
                                        /* loop will allow incoming cmds */

#endif  /* CCS_FULL || CCS_LIGHT */

/*
 * Data types
 */

/*
```

```
 * Function prototypes
 */
int Seq_Init (                  /* Sequencer interpreter initialization */
        Tcl_Interp *interp      /* pointer to interpreter structure */
        );                      /* (called by Tcl_AppInit & seqInitInterp) */

#if defined(CCS_FULL) || defined(CCS_LIGHT)
EXTERN ccsCOMPL_STAT seqInitInterp(  /* creates/initializes a Sequencer interp */
        char       *display,    /* string with display-term name */
        char       *name,       /* label for main window */
        vltLOGICAL interactive, /* flags if application is shell-like */
        Tcl_Interp **interp,    /* Tcl interpreter structure pointer-pointer */
        ccsERROR   *error);

EXTERN ccsCOMPL_STAT seqEval (  /* Sequencer interface to Tcl_Eval */
        Tcl_Interp *interp,     /* Tcl interpreter pointer */
        char       *script,     /* string with script */
        ccsERROR   *error);     /* error structure pointer */

EXTERN ccsCOMPL_STAT seqMonitorQ ( /* enables/disables monitoring of msg queue */
        Tcl_Interp *interp,     /* result string goes to interp->result */
        vltLOGICAL stateflag,   /* ccsTRUE => enable; ccsFALSE => disable */
        ccsERROR   *error);     /* returned error structure */

#endif  /* CCS_FULL || CCS_LIGHT */
EXTERN ccsCOMPL_STAT saveAndCloseStack (/* does a errCloseStack after saving */
        ccsERROR   *error);     /* this error structure in global lastError */

#endif  /*!SEQ_H*/
```